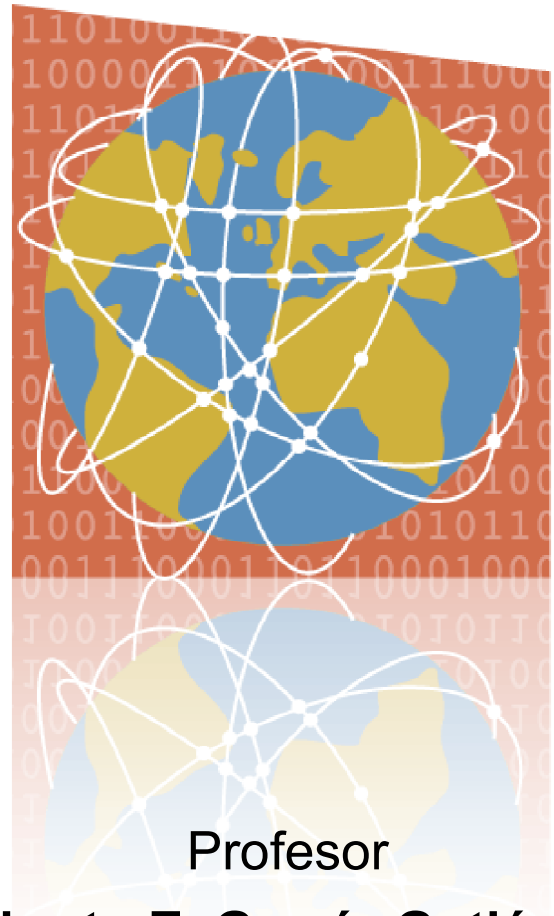
# Protocolos de Interconexión de Redes

## Tema III

## El Nivel de Transporte



Profesor

**Alberto E. García Gutiérrez**

# Tema III

# EL NIVEL DE TRANSPORTE

## *INTRODUCCIÓN*

Se encarga este nivel de suministrar el servicio de transporte de bits a las aplicaciones que se comunican. Éstas funcionan generalmente de acuerdo con el paradigma cliente-servidor, por el cual una aplicación (cliente) toma la iniciativa y solicita los servicios a la otra (servidor). Como ya sabemos la comunicación 'peer to peer' entre dos entidades del nivel de transporte ocurre en realidad gracias a los servicios ofrecidos por el nivel de red. Mientras que el nivel de red se ocupa de resolver los problemas propios de la topología de ésta (rutas y congestión fundamentalmente) el nivel de transporte sólo existe en las dos entidades extremas de la comunicación, por lo que también se le llama nivel host-host. El nivel de transporte no es consciente, ni debe serlo, de la manera como físicamente están interconectados los dos hosts, que puede ser por una LAN, una WAN o una combinación de múltiples redes de ambos tipos.

La unidad básica de información a nivel de enlace se denomina *trama* (porque los datos van 'rodeados' de información de control por delante y por detrás). En el nivel de red esta unidad básica se conoce como *paquet*e. No existe un término equivalente para la unidad de transferencia de información en el nivel de transporte; a falta de mejor alternativa utilizaremos el término *TPDU* (Transport Protocol Data Unit) para indicarla; en la Internet se suele utilizar el término *mensaje* en el caso de UDP (servicio no orientado a conexión), y *segmento* en el de TCP (servicio orientado a conexión), pero esta nomenclatura no es compartida por otros protocolos de transporte.

Generalmente las aplicaciones requieren que el nivel de transporte les garantice la entrega de los datos al destinatario, sin errores, pérdidas ni datos duplicados; para que esto sea posible el nivel de transporte ofrecerá normalmente un servicio orientado a conexión, con retransmisiones en caso necesario. Este es el caso por ejemplo del protocolo TCP de Internet, utilizado en muchas aplicaciones como FTP (File Transfer Protocol, transferencia de ficheros), SMTP (Simple Mail Transfer Protocol, correo electrónico) , HTTP (HyperText Transfer Protocol, usado en tráfico Web), etc.

En ocasiones las aplicaciones prefieren un servicio menos fiable en el que los mensajes sean enviados sin pedir confirmación, de forma independiente unos de otros. Este tipo de servicio se suministra normalmente con un protocolo no orientado a conexión. Existen diversas razones que pueden justificar este tipo de servicios, por ejemplo cuando se envía información en tiempo real. El protocolo UDP de Internet es un ejemplo de este tipo de servicio.

En OSI también hay dos protocolos de transporte, uno orientado a conexión y uno no orientado a conexión. En la siguiente tabla mostramos los más importantes de cada grupo, incluyendo los de nivel de red:

| Tipo de protocolo | Internet | OSI |
|---|---|---|
| Nivel de red | IP (Internet Protocol) | CLNP (ConnectionLess Network Protocol) |
| Routing interno | OSPF (Open Shortest Path First) | IS-IS ( Intermediate System to Intermediate System) |
| Routing externo | BGP (Border Gateway Protocol) | IDRP (InterDomain Routing Protocol) |
| Nivel de transporte, orientado a conexión | TCP (Transmission Control Protocol) | TP4 (Transport Protocol clase 4) |
| Nivel de transporte, no orientado a conexión | UDP (User Datagram Protocol) | TP0 (Transport Protocol clase 0) |

Conviene recordar que en el modelo OSI el nivel de transporte presta sus servicios al nivel de sesión, y no directamente al de aplicación.

Normalmente las entidades del nivel de transporte se implementan como procesos en el sistema operativo del host, o bien procesos de usuario. El sistema operativo puede ser monousuario o multiusuario. En muchos casos el host tiene una sola instancia del nivel de red, y una sola del de transporte, pero muchas de los niveles superiores (aplicación o sesión); el nivel de transporte se encarga de mutiplexar el tráfico recibido de las diversas entidades de nivel superior en una única conexión con el nivel de red.

Los protocolos de transporte no orientados a conexión, como UDP, son protocolos muy sencillos, que existen únicamente para permitir la correcta conexión de la capa de aplicación y la de red; actúan como una capa de adaptación bastante primitiva. Por esto la mayoría de nuestra discusión se centrará en los protocolos orientados a conexión, e implícitamente nos referiremos a ellos la mayor parte del tiempo.

Uno de los aspectos que suelen establecerse a nivel de transporte es la calidad de servicio (QoS) que se desea. Esta se especifica en el momento de establecer la conexión, y tendrá su efecto en el nivel de red. Por ejemplo una determinada aplicación en Internet requiere establecer una conexión con una elevada fiabilidad; como consecuencia de esto TCP solicita a IP la inclusión del bit correspondiente en el campo TOS (Type Of Service) en todos los datagramas correspondientes a esta conexión.

Otros parámetros del nivel de red que pueden verse influidos por el nivel de transporte en Internet son el campo precedencia (prioridad) o la reserva de recursos de RSVP. En algunos protocolos se contempla el uso de múltiples parámetros para definir la calidad de servicio en el momento de establecer la conexión; si la calidad solicitada no puede cumplirse, por no haber recursos suficientes, la red puede proponer una contraoferta que el solicitante puede o no aceptar; este proceso se conoce como negociación de opciones.

Los parámetros más habituales para fijar la calidad de servicio son los siguientes:

- *Retardo en el establecimiento de la conexión:* tiempo que transcurre desde que el nivel de transporte realiza la solicitud hasta que recibe la confirmación desde el otro extremo de que ha sido efectuada.
- *Probabilidad de fallo en el establecimiento de la conexió*n: la conexión puede no establecerse dentro del tiempo máximo previsto por algún problema en la red, por ejemplo congestión, avería, etc.

- *Rendimiento*: mide la cantidad de información transmitida por unidad de tiempo, normalmente en bytes por segundo. Depende de la velocidad de las líneas utilizadas, de su grado de saturación, del tipo de protocolo utilizado (p. ej. ventana deslizante) y de los hosts utilizados.
- *Retardo de tránsito*: el tiempo que tarda la TPDU en llegar al host de destino (equivale al retardo o latencia del nivel de red o el de enlace, pero en este caso medido entre las entidades en el nivel de transporte).
- *Tasa de error residual*: mide la proporción de TPDUs perdidas o erróneas. En teoría en un servicio fiable (como TCP) esta debiera ser cero, pero en la práctica puede tener un valor mayor que cero, aunque muy pequeño.
- *Protección*: permite al usuario solicitar protección contra accesos no autorizados por parte de terceros a la información que envía. Algunos protocolos prevén el uso de técnicas criptográficas en estos casos.
- *Prioridad*: permite especificar niveles de importancia del tráfico enviado; en caso de congestión la red descartará primero el tráfico menos prioritario. Este campo puede tener consecuencias por ejemplo en el bit DE de Frame Relay o el bit CLP de ATM; también en el campo TOS (Type Of Service, precedencia y bits DTRC) del datagrama IP.
- *Resistencia*: indica la probabilidad de que la conexión sea abortada de forma anormal por el nivel de transporte debido a problemas en la red (excesiva congestión, por ejemplo).

## *ELEMENTOS DE PROTOCOLOS DE TRANSPORTE*

Al ocuparse de la comunicación extremo a extremo o punto a punto, el nivel de transporte se parece en algunos aspectos al nivel de enlace. Así por ejemplo, entre los asuntos de que normalmente habrá de ocuparse se encuentran el control de errores (incluyendo mensajes perdidos o duplicados) y el control de flujo. Aunque las técnicas que se aplican son parecidas, existen importantes diferencias entre ambos motivadas por el hecho de que en el nivel de enlace hay sólo un hilo físico (o su equivalente) entre las dos entidades comunicantes, mientras que en el nivel de transporte hay toda una red. Las mayores diferencias entre el nivel de transporte y el de enlace son las siguientes:

- El retardo que se observa en el nivel de transporte es normalmente mucho mayor y sobre todo más variable (mayor jitter) que en el de enlace.
- En el nivel de enlace el medio físico entre las dos entidades tiene una capacidad normalmente muy reducida, y siempre la misma, de almacenar información; en el de transporte los routers intermedios pueden tener una capacidad considerable de almacenamiento de la información, y esta capacidad puede variar con el estado de la red.
- En el nivel de enlace se asegura que las tramas llegarán al receptor en el mismo orden que han salido del emisor (salvo que se pierdan, en cuyo caso no llegarán); en el nivel de transporte esto es cierto solo cuando se utiliza un servicio orientado a conexión en el nivel de red; si se utiliza un servicio no orientado a conexión el receptor podría recibir los datos en orden distinto al de emisión.
- En el nivel de enlace las dos entidades se 'ven' directamente (suponiendo una comunicación dúplex); a veces incluso de forma permanente (por ejemplo en una comunicación síncrona DIC están continuamente emitiendo la secuencia

01111110); esto permite que el emisor sepa en todo momento si el receptor está operativo, y el receptor sabe que los datos recibidos corresponden todos a una misma sesión del emisor. En el nivel de transporte la comunicación es indirecta; el emisor podría enviar datos, quedar fuera de servicio y más tarde entrar en funcionamiento otra vez; si no se adoptan las medidas oportunas el receptor podría recibir todos esos datos sin siquiera percatarse de que corresponden a dos sesiones distintas del emisor (incluso a dos usuarios distintos).

En el nivel de enlace no es necesario normalmente incluir la dirección del destinatario, pues solo hay una posibilidad (salvo en las redes broadcast). En el nivel de transporte siempre se ha de incluir el destinatario, pues hay muchos posibles interlocutores.

Recordemos que en el modelo OSI cada nivel presta sus servicios al nivel superior a través del SAP (Service Access Point), cada uno de los cuales es identificado por una dirección. Por ejemplo en Internet la dirección SAP por la que el nivel de red accede al servicio es la dirección IP del host, que hemos visto en el tema anterior. La dirección SAP por la que el nivel de transporte accede al servicio de red está formada por el campo *protocolo* del datagrama IP (recodemos que entre los valores posibles de dicho campo estaba por ejemplo 6 para TCP y 17 para UDP). A su vez el nivel de transporte ofrece sus servicios al nivel de aplicación a través de unos SAPs específicos, que en el caso de Internet son los denominados *ports* o *puerto*s. Estos puertos se denominan también TSAPs (Transport SAPs).

Para que un proceso cliente (o aplicación) de un host pueda comunicar con un servidor haciendo uso de los servicios de su nivel de transporte es preciso que conozca el TSAP correspondiente.

Normalmente el TSAP del servidor forma parte del estándar del protocolo, por lo que es universalmente conocido y cualquier cliente que lo desee sabe que TSAP debe utilizar para acceder a dicho servidor. En cambio el TSAP del cliente no necesita ser conocido por otros usuarios.

### Establecimiento de una conexión

En principio para establecer una conexión el cliente emite una TPDU de petición de conexión, y el servidor responde con una TPDU de aceptación; a partir de ese momento puede empezar el intercambio de datos. Sin embargo cuando analizamos mas a fondo las posibles anomalías empiezan a surgir los problemas, que pueden ser muchos.

Recordemos que en el nivel de transporte puede haber una gran fluctuación (a veces del orden de segundos) en el tiempo que tardan en llegar las TPDUs a su destino; las TPDUs pueden perderse o llegar duplicadas (ya que si el emisor no recibe confirmación reenviará la misma TPDU pasado el timeout). Imaginemos que el cliente intercambia una serie de TPDUs con el servidor, y cuando ya ha terminado su transacción cierra la sesión; segundos mas tarde de algún rincón de la red aparecen las mismas TPDUs del cliente duplicadas que llegan al servidor de nuevo; éste realizaría la misma transacción otra vez, con efectos posiblemente desastrosos.

Para evitar esto se utiliza al establecer la conexión el mecanismo conocido como *saludo*

*de tres vía*s, (three-way handshake). La idea es que el servidor sólo aceptará la conexión después de haber pedido al cliente confirmación de que desea realizarla. En principio esto por sí solo no resuelve nuestro problema, ya que cabría pensar que después de la TPDU de petición inicial duplicada la red le entregara al servidor la TPDU de confirmación, también retrasada.

La solución a este problema se obtiene por el siguiente mecanismo. Tanto el cliente como el servidor utilizan un protocolo de ventana deslizante para el envío de las TPDUs, para lo cual han de emplear un número de secuencia; a diferencia del número de secuencia que vimos en el nivel de enlace, el del nivel de transporte emplea rangos muy grandes (por ejemplo en TCP el número de secuencia es módulo 2 32 ); tanto el cliente como el servidor eligen de forma aleatoria o pseudoaleatoria el valor inicial del número de secuencia que van a utilizar, cada uno por separado para cada sentido de la comunicación. El cliente informa al servidor en su primera TPDU del número de secuencia elegido, y éste le responde en su TPDU con el número de secuencia elegido por él, y acompañará un ACK piggybacked de la TPDU recibida. De esta forma si el servidor recibe una TPDU de petición de conexión vieja responderá con una TPDU al cliente en la que pondrá en el campo ACK el número de secuencia recibido; el cliente verá que ese número no corresponde con ninguna conexión que él tuviera pendiente de confirmación, por lo que rechazará la conexión; el servidor por su parte esperará recibir en el campo ACK de la siguiente TPDU un valor que corresponda con el que él ha enviado en la anterior.

La técnica de los números de secuencia aleatorios evita también el riesgo de que un cliente caiga (por ejemplo por un fallo de corriente) y luego al levantarse de nuevo intente utilizar la misma conexión; al no coincidir el número de secuencia con el que estaba vigente, el nuevo proceso cliente no podrá utilizar la vieja conexión. Esta es también una medida de seguridad ya que el nuevo proceso cliente podría ser un impostor (imaginemos que durante la sesión anterior se ha realizado una identificación con clave usuario/password, el nuevo cliente podría acceder al servidor sin necesidad de identificarse de nuevo).

Generalmente se establece una vida máxima para las TPDUs en la red; de esta forma se reduce el riesgo de recibir duplicados retrasados. Cuando un sistema cae y vuelve a arrancar se recomienda esperar a que pase el tiempo de vida de las TPDUs antes de activar el nivel de transporte; de esta manera es imposible que una TPDU de la sesión anterior pueda aun aparecer por alguna parte cuando se inicia la sesión nueva. En Internet por ejemplo el tiempo de vida recomendado de las TPDUs es de 2 minutos, y se controla a partir del campo TTL en el datagrama IP.

Una vez establecidos los números de secuencia es posible utilizar para el intercambio de TPDUs cualquier protocolo de ventana deslizante, tanto con retroceso n como con repetición selectiva. A diferencia del nivel de enlace, donde el protocolo se basaba en numerar las tramas, en el nivel de transporte se suele numerar cada byte, ya que el tamaño de las TPDUs puede ser muy variable. Para las retransmisiones se puede utilizar tanto retroceso n como repetición selectiva.

## Terminación de una conexión

Una conexión puede terminarse de forma simétrica o asimétrica. La terminación asimétrica es unilateral, es decir uno de los dos hosts decide terminar y termina la conexión en ambos sentidos. En la terminación simétrica cada host corta la conexión únicamente en el sentido que emite datos; podemos considerar la terminación simétrica como dos circuitos simplex donde cada uno es controlado por el emisor.

La terminación asimétrica se considera anormal y puede provocar la pérdida de información, ya que cuando un host ha enviado la TPDU de desconexión ya no acepta más datos; entretanto el otro host podría haber enviado una TPDU de datos que no será aceptada.

En el caso más normal de la terminación simétrica el host 1 'invita' al host 2 a desconectar mediante una TPDU DISCONNECT REQUEST; el host 2 responde con otra DISCONNECT REQUEST, a la cual el host 1 responde con una TPDU ACK y cierra la conexión; el host 2 cerrará la conexión al recibir el ACK. Por este mecanismo se asegura que no se pierden TPDUs 'en ruta' ya que ambos hosts tienen aviso previo de la desconexión y dan su conformidad explícitamente. Obsérvese que este mecanismo supone el intercambio de tres mensajes de forma análoga al proceso de conexión, por lo que también se denomina saludo a tres vías; no existe forma fiable de terminar la conexión en menos mensajes sin correr el riesgo de perder datos.

Si se pierde alguna de las TPDUs de desconexión el mecanismo fallará pues los hosts se quedarán esperando eternamente la respuesta. Para evitar esto se utiliza un mecanismo de timeouts que resuelve el problema reenviando la TPDU perdida si se trata de un DISCONNECT REQUEST, o cerrando la conexión por timeout cuando lo que se ha perdido es el ACK. En el Tanenbaum Fig. 6-14 aparece una relación de los casos 'patológicos' posibles y como se resuelven.

Existen muchas circunstancias que pueden provocar que una conexión se quede medio abierta, es decir abierta sólo por un lado. Por ejemplo, un host puede quedar fuera de servicio sin previo aviso y el otro, que tenía una conexión abierta con él, quedar a la espera sin saber nada de lo ocurrido. Para resolver estas situaciones se prevé normalmente un tiempo máximo que una conexión puede estar abierta sin tráfico; pasado ese tiempo los hosts se enviarán mensajes de prueba (denominados keep-alive en TCP) para comprobar que el otro lado aún responde. Los valores de timeout para el envío de mensajes keep-alive son increíblemente grandes, la documentación de TCP sugiere 2 horas como valor por defecto. Un valor muy pequeño podría provocar que un fallo momentáneo en la red cerrara conexiones a nivel de transporte, perdiendo así la principal ventaja de las redes de datagramas.

Analicemos ahora que ocurre si dos hosts tienen establecida una conexión entre ellos y falla la red que los une. En el caso de utilizar un servicio orientado a conexión (X.25, ATM) generalmente la sesión termina de manera abrupta, pues la vía de comunicación (el circuito virtual) ha desaparecido y para restaurar la comunicación habrá que restablecer el circuito, presumiblemente por un camino físico diferente. En el caso de utilizar un servicio de red no orientado a conexión (IP, OSI CLNP) la red reencaminará los datagramas por una ruta alternativa (suponiendo que ésta exista) por lo que lo único que el nivel de transporte detectará es la pérdida de unas pocas TPDUs, pero la

conexión no se cortará.

### Control de flujo y de buffers

El control de flujo en el nivel de transporte es fundamental, ya que la velocidad con que los datos llegan al receptor puede ser muy variable al intervenir multitud de factores. Como ya hemos dicho se suelen utilizar protocolos de ventana deslizante. Mientras que en el nivel de enlace se asignaba de manera estática un espacio de buffers a cada conexión, en el nivel de transporte esta estrategia no es adecuada, pues el número de conexiones simultáneas puede variar muchísimo, al no haber una interfaz física asociada a cada conexión.

Por este motivo la asignación de espacio para buffers en el nivel de transporte tiene dos características singulares que le diferencian del nivel de enlace. En primer lugar el espacio de buffers es común y compartido por todas las conexiones, entrantes y salientes. En segundo lugar el reparto del espacio entre las conexiones activas se hace de forma dinámica en función de las necesidades; una conexión con poco tráfico recibirá menos asignación que una con mucho tráfico. En todo momento cada conexión tiene asignado un espacio para emisión y uno para recepción; el espacio de emisión está ocupado con TPDUs pendientes de confirmación o de ser enviadas al otro host; el espacio de recepción tiene una parte ocupada con TPDUs ya recibidas pendientes de ser aceptadas por el nivel de aplicación, y otra reservada para TPDUs que pueden llegar del otro host. Otra diferencia respecto al nivel de enlace estriba en que el tamaño de las TPDUs puede ser muy variable (mientras que el tamaño de las tramas suele ser mas constante para una conexión física dada). Para optimizar la utilización del espacio se asignan segmentos de buffer de longitud variable.

Como consecuencia de esto en el nivel de transporte tanto los números de secuencia como los tamaños de ventana cuentan generalmente bytes, no TPDUs.

La parte de buffer que el receptor tiene reservada para TPDUs que puedan llegarle es anunciada al emisor regularmente, para que éste sepa que cantidad de datos esta dispuesto a aceptar el receptor.

Este espacio puede fluctuar mucho con el tiempo en función de la actividad de esa y de las demás conexiones que mantenga el host.

Obsérvese que con este modo de funcionamiento el receptor controla la situación, ya que si indica una ventana cero el emisor tendrá que esperar y no enviarle datos mientras el receptor no le anuncie una ventana mayor.

En redes no orientadas a conexión los datagramas (y por tanto las TPDUs) pueden llegar desordenados, por lo que el nivel de transporte debe estar preparado para recibir números de secuencia desordenados (siempre y cuando estén dentro de la ventana establecida).

### Multiplexación

En las redes públicas de conmutación de paquetes (X.25, frame relay y ATM), que son

orientadas a conexión, el usuario paga por cada circuito virtual, lo cual estimula a utilizar el mínimo número de circuitos posible. Generalmente es el nivel de transporte el encargado en estos casos de multiplexar la diferentes conexiones solicitadas por el nivel de aplicación en una única conexión a nivel de red; dicho en terminología OSI el nivel de transporte presenta diferentes TSAPs sobre un único NSAP.

Esto se conoce como multiplexación hacia arriba, ya que visto en el modelo de capas supone que varias direcciones del nivel de transporte confluyan en una única dirección del nivel de red.

También en redes no orientadas a conexión (IP o ISO CLNP) el nivel de transporte suele ocuparse de multiplexar el tráfico de las diferentes aplicaciones y usuarios (cada aplicación puede estar siendo utilizada por varios usuarios) en una única dirección a nivel de red.

Existen otras situaciones en las que interesa hacer multiplexación en sentido opuesto. Supongamos por ejemplo el caso de un servidor al que para mejorar su rendimiento se le han instalado dos interfaces de red, por ejemplo dos controladores Ethernet; supongamos que el servidor utiliza TCP/IP, y posee por tanto una dirección IP para cada interfaz. El servidor esta dedicado a una única aplicación, por lo que utiliza un único puerto. En este caso necesitamos hacer multiplexación hacia abajo. En ocasiones, debido a la forma como funcionan los protocolos o como se reparte la capacidad disponible, la multiplexación hacia abajo es interesante incluso cuando hay una sola interfaz física con la red; por ejemplo, si el protocolo a nivel de transporte no utiliza ventana deslizante sino parada y espera el crear varias conexiones a nivel de transporte para un mismo usuario del nivel de aplicación permite aprovechar los tiempos de espera que produce el protocolo. También sería útil la multiplexación si el algoritmo de reparto de recursos se basa en los usuarios del nivel de transporte; presentando varias conexiones obtendremos mejor rendimiento (seguramente con detrimento de los demás usuarios).

### Recuperación de caídas

Ya hemos visto el tipo de medidas preventivas que se adoptan para evitar que cuando una instancia del nivel de transporte en un host cae y se levanta mas tarde no sea posible recuperar la conexión previamente establecida, y haya que crear una nueva. Esto es una medida de seguridad fundamental para evitar inconsistencias en la información y accesos no autorizados.

Otro problema importante es que ocurre cuando cae todo un host, lo cual provoca la caída simultánea del nivel de transporte y el nivel de aplicación. Supongamos por ejemplo que un cliente está realizando una serie de actualizaciones en una base de datos, cada una de ellas contenida en una TPDU; a cada transacción el servidor responde con un ACK indicando que ha efectuado la operación correspondiente. En un determinado momento el servidor cae, rearrancando a continuación; podemos concluir que si el cliente ha recibido el ACK es que la actualización se ha efectuado, y si no no, pero como la actualización y el envío del ACK son sucesos consecutivos y no simultáneos siempre ocurrirá uno primero y el otro después; cualquiera que sea el orden elegido siempre podrá ocurrir que el host caiga entre ambos eventos, con lo que tendremos o

bien una actualización efectuada y no notificada, o una notificación enviada al cliente de una actualización no realizada. Este tipo de problemas solo puede resolverse a nivel de aplicación mediante una posterior verificación de lo que realmente ha ocurrido.

## *TCP : PROTOCOLO DE CONTROL DE TRANSMISIÓN*

El servicio que proporciona TCP al nivel de aplicación es completamente diferente del que proporciona UDP, si bien ambos hacen uso del mismo protocolo de red: IP. TCP proporciona un servicio de transmisión orientado a la conexión y fiable, considerando toda la información intercambiada durante una conexión como un flujo continuo de bytes.

El hecho de que el servicio sea orientado a la conexión supone que las dos aplicaciones que utilizan TCP para comunicarse (normalmente un cliente y un servidor) deben establecer una conexión TCP entre ellas antes de poder intercambiar datos.

Los datos que la aplicación entrega a TCP se dividen en bloques del tamaño que TCP considera adecuado para ser entregado a IP; a cada una de estas unidades de información se le denomina **segment**o. Este funcionamiento es completamente diferente al de UDP, donde cada escritura de la aplicación genera un datagrama UDP del mismo tamaño a la unidad de información.

Cuando TCP envía un segmento arranca un **temporizado**r, esperando que el otro extremo de la comunicación reconozca la recepción de dicho segmento. Cuando TCP recibe datos procedentes del otro extremo de la conexión, envía un reconocimiento confirmándolos, si bien no de forma inmediata sino retrasándolo una fracción de segundo. Cuando TCP no recibe un **reconocimiento** positivo de un segmento enviado en un tiempo determinado se produce un "timeout" del temporizador y el segmento será **retransmitid**o.

Con el fin de poder detectar cualquier modificación de los datos durante su transmisión, TCP calcula un **Checksum** que incluye en la cabecera y que verifica la integridad del segmento. Si un segmento llega con un Checksum no válido, TCP lo descarta, no enviando un reconocimiento de su recepción.

Puesto que los segmentos TCP se transmiten en forma de datagramas IP, y dado que los datagramas IP pueden llegar desordenados, los segmentos TCP pueden recibirse desordenados. El protocolo TCP receptor **reordena** los datos, si es necesario, entregándolos a la aplicación en el orden correcto.

Dado que los datagramas IP pueden duplicarse, el protocolo TCP receptor debe **eliminar los datos duplicado**s.

TCP proporciona también un **control de fluj**o. Cada uno de los extremos de una conexión TCP dispone de un espacio de almacenamiento finito para los segmentos recibidos. El protocolo TCP receptor sólo permite al otro extremo enviar tantos datos como pueda almacenar en su memoria; de esta forma, evita que un emisor muy rápido pueda agotar la memoria de almacenamiento de un receptor más lento.

La aplicaciones de los extremos de una conexión TCP intercambian entre sí un flujo continuo de octetos de información, de modo que aunque una aplicación en uno de los extremos escriba varias secuencias la aplicación del otro extremo no puede discernir cuáles fueron las escrituras individuales. Cada uno de los extremos de la conexión transmite un flujo de octetos y TCP no interpreta el significado de los mismos ( no tiene idea de si los datos que están siendo intercambiados son datos binarios, caracteres ASCI, etc ). La interpretación de este flujo de octetos corresponde a las aplicaciones y no al protocolo TCP.

Al igual que UDP, TCP hace uso de los **número de puerto de protocolo** para identificar una aplicación dentro del sistema de destino. TCP se fundamenta en el concepto de **conexión** entre aplicaciones. Una conexión entre dos aplicaciones se identifica por sus **extremos;** un extremo es un par ( nodo, puerto ) que identifica respectivamente el sistema y la aplicación que constituyen cada uno de los extremos. Un puerto TCP puede ser empleado simultáneamente por varias conexiones. Así pues, una conexión de dos entidades usuarias del nivel de transporte se especifica por la combinación:

Dir. IP Host 1 + port Host 1 + Dir. IP Host 2 + port Host 2

El número de Puerto es un número entero entre 0 y 65535. Por convenio los números 0 a 1023 están reservados para el uso de servicios estándar, por lo que se les denomina **puertos bien conocidos** (*well-known ports*). Cualquier número por encima de 1023 está disponible para ser utilizado libremente por los usuarios. Los valores de los puertos bien conocidos se encuentran recogidos en la **RFC 1700** (*Assigned Internet Numbers*):

| Puerto | Aplicación | Descripción |
|---|---|---|
| 9 | Discard | Descarta todos los datos recibidos (para pruebas) |
| 19 | Chargen | Intercambia cadenas de caracteres (para pruebas) |
| 20 | FTP-Data | Transferencia de datos FTP |
| 21 | FTP | Diálogo en transferencia de ficheros |
| 23 | TELNET | Logon remoto |
| 25 | SMTP | Correo electrónico |
| 110 | POP3 | Servidor de correo |
| 119 | NNTP | News |

Por ejemplo, supongamos que cinco usuarios del host de dirección IP 134.123.1.2 inician una sesión de logon remoto hacia el host de dirección 221.198.34.21; cada uno de ellos ejecutará en su host un programa telnet cliente que abrirá una conexión con el otro; las conexiones establecidas podrían ser por ejemplo:

```
(134.123.1.2, 1024) con (221.198.34.21, 23)
(134.123.1.2, 1025) con (221.198.34.21, 23)
(134.123.1.2, 1026) con (221.198.34.21, 23)
(134.123.1.2, 1030) con (221.198.34.21, 23)
(134.123.1.2, 1031) con (221.198.34.21, 23)
```

donde hemos empleado la notación (dirección IP, puerto). Obsérvese que la asignación de puertos para los clientes se hace por simple orden de llagada a partir del primer número de puerto no reservado. En el servidor todas las conexiones utilizan el puerto 23

(pues todas acceden al mismo proceso, el servidor telnet); en cambio en el cliente cada usuario es un proceso diferente y utiliza un puerto distinto. La combinación (dirección IP, puerto) se denomina SOCKET (igual que las APIs utilizadas en UNÍS para acceder a los servicios TCP).

Complicando aun más nuestro ejemplo anterior podríamos imaginar que el host cliente estuviera "multihomed", es decir que tuviera por ejemplo dos interfaces físicas, y por tanto tuviera dos direcciones IP; los usuarios podrían utilizar ambas interfaces alternativamente; por lo que las conexiones podrían ser por ejemplo:

```
(134.123.1.2, 1024) con (221.198.34.21, 23)
(134.123.1.3, 1024) con (221.198.34.21, 23)
(134.123.1.2, 1025) con (221.198.34.21, 23)
(134.123.1.3, 1025) con (221.198.34.21, 23)
(134.123.1.2, 1030) con (221.198.34.21, 23)
```

Por último, el host cliente podría simultáneamente a las sesiones telnet enviar datagramas UDP al host B; aquí, aunque no se establece una conexión (pues de trata de un servicio no orientado a la conexión CNLS) hay un puerto de origen y uno de destino; los datagramas podrían tener como puerto de origen el 1024 y de destino el 23; esto no causaría ninguna ambigüedad, ya que el campo protocolo del datagrama IP diferenciaría ambos tipos de paquetes, y los entregaría al servicio correspondiente en el nivel de transporte del receptor,

En resumen, el protocolo TCP define los siguientes aspectos :

- Formato de los datos y de los reconocimientos.
- Procedimientos para verificar que los datos lleguen correctos.
- Esquema para distinguir entre distintos destinos en un sistema.
- Procedimientos de recuperación de datos perdidos o corrompidos.
- Procedimiento de inicio y liberación de la conexión.

## FORMATO DE LOS SEGMENTOS TCP

La unidad de transferencia en TCP es el **segment**o. Se utilizan segmentos para :

- Establecer conexiones
- Transferir datos
- Enviar reconocimientos
- Notificar modificaciones del tamaño de la ventana
- Cerrar conexiones

Sin embargo, el formato de los segmentos es en todos los casos el mismo, tal y como se muestra en la figura:

<table>
<tr><td colspan="4">0 4 10 16 24 31</td></tr>
<tr><td colspan="3">PUERTO DE ORIGEN</td><td>PUERTO DE DESTINO</td></tr>
<tr><td colspan="4">NÚMERO DE SECUENCIA DE ENVÍO</td></tr>
<tr><td colspan="4">NÚMERO DE RECONOCIMIENTO</td></tr>
<tr><td>LONGITUD CABECERA</td><td>RESERVADO</td><td>FLAGS</td><td>TAMAÑO DE VENTANA</td></tr>
<tr><td colspan="3">CHECKSUM</td><td>PUNTERO URGENTE</td></tr>
<tr><td colspan="3">OPCIONES ( SI LAS HAY )</td><td>RELLENO</td></tr>
<tr><td colspan="4">DATOS</td></tr>
<tr><td colspan="4">. . .</td></tr>
</table>

El tamaño habitual de la cabecera TCP es de 20 octetos, a menos que esté presente el campo opciones.

Cada segmento TCP contiene los **números de puerto de origen y de destino** para identificar las aplicaciones emisora y receptora. Estos dos valores, junto con la dirección IP de destino y origen de la cabecera IP identifican de forma unívoca cada conexión.

La combinación de la dirección IP y del número de puerto se denomina generalmente "Socket". Este término apareció en la especificación original de TCP y posteriormente se empleó como el nombre del Interfaces de programación derivado del Unix de Berkeley. Un par de Sockets ( dirección IP y número de puerto del cliente y dirección IP y número de puerto del servidor ) especifican los dos extremos que identifican unívocamente cada una de las conexiones de una red.

El campo **número de secuencia** identifica la posición que ocupa el primer octeto de datos de cada segmento en la secuencia de datos correspondiente a una conexión. TCP asigna un número de secuencia a cada octeto transmitido entre dos aplicaciones en un sentido. Este número de secuencia es un número de 32 bits sin signo que vuelve a 0 después de alcanzar el valor $2^{32}$ -1.

Cuando se establece una nueva conexión mediante un segmento, el flag **SYN** está activado, tal y como veremos más adelante. En este primer segmento de la conexión, el campo **número de secuencia** contiene el **número de secuencia inicial** (NSI) elegido por este nodo para esta conexión.

El número de secuencia del primer octeto de datos enviado por este nodo tendrá el valor : NSI + 1, dado que el envío del flag SYN consume un número de secuencia.

Dado que cada octeto que se intercambia es numerado, el campo **número de reconocimiento** contiene el número de secuencia del siguiente octeto que el emisor de un reconocimiento espera recibir. Corresponde, por lo tanto, al número de secuencia más uno del último octeto de datos recibido correctamente. Este campo sólo es válido si el flag **ACK** está activado.

El envío de reconocimientos ( ACK ) no supone ninguna sobrecarga en la red, dado que el campo número de reconocimiento de 32 bits forma parte siempre de la cabecera, al igual que el flag ACK. Es por ello que una vez que se ha establecido una conexión, este

campo siempre se utiliza y el flag ACK siempre está activado.

TCP proporciona al nivel de aplicación un servicio full dúplex, esto significa que los datos pueden circular en cada sentido independientemente. Por lo tanto, cada extremo de una conexión debe mantener un número de secuencia de los datos que envía.

TCP es un protocolo que utiliza un mecanismo de ventana deslizante sin reconocimientos selectivos ni negativos.

El campo **longitud de cabecera** especifica la longitud de la cabecera del segmento medida en palabras de 32 bits, este campo es necesario dado que la longitud de la cabecera es variable en función del campo opciones. Dado que se emplea un campo de 4 bits para especificar su longitud, la cabecera TCP está limitada a 60 octetos. Sin embargo, sin el campo opciones, el tamaño normal de la cabecera es de 20 octetos.

La cabecera TCP contiene 6 flags que pueden ser activados o no de forma independiente, y cuyo significado determina el propósito y el contenido del segmento. Estos bits indican el modo de interpretar algunos de los campos de la cabecera de acuerdo con la tabla siguiente:

- URG (Urgent) El campo de datos contiene información urgente y se ha hecho uso del campo **puntero urgent**e, que es el que contiene la dirección donde terminan estos.
- ACK (Acknowledgement) Se ha hecho uso del campo **número de reconocimiento.** En la práctica el bit ACK está a 1 siempre, excepto en el primer segmento enviado por el host que inicia la conexión.
- PSH (Push) El receptor debe pasar estos datos a la aplicación tan pronto como le sea posible, sin esperar a acumular varios segmentos.
- RST (Reset) Debe reinicializarse la conexión. Indica que se ha detectado un error de cualquier tipo, por ejemplo, una terminación unilateral de una conexión o que se ha recibido un segmento con un valor inadecuado del número de secuencia o número de ACK.
- SYN (Synchronize) Petición de sincronismo de números de secuencia para iniciar la conexión. El campo **número de envío** contiene el **número inicial de secuencia.** Este flag solo está puesto en el primer mensaje enviado por cada lado en el inicio de la conexión.
- FIN (Finish) El emisor ha terminado de enviar datos. Para que una conexión se cierre de manera normal cada host ha de enviar un segmento con el bit FIN puesto.

El control de flujo de TCP se efectúa mediante un mecanismo de ventana deslizante. A diferencia de otros protocolos que emplean este mismo mecanismo, el tamaño de la ventana es variable, lo que se consigue mediante el campo **tamaño de ventan**a. Este campo contiene el número de octetos, comenzando con el especificado por el campo **número de reconocimiento** que el receptor está dispuesto a aceptar. Este campo de 16 bits limita el tamaño de la ventana a 65535 octetos. Como veremos más adelante, la opción **escalado de ventana** permite escalar este valor para conseguir utilizar ventanas mayores.

El campo **checksum** sirve para comprobar la corrección del segmento TCP: la cabecera y los datos. Es un campo obligatorio, que debe calcular e incluir en la cabecera el protocolo TCP emisor para después ser verificado por el receptor. El checksum TCP se calcula de forma similar al checksum UDP, utilizando una pseudocabecera tal y como se describe en el apartado correspondiente.

| 0 | 8<br>31 | 16 |
|---|---|---|
| DIRECCIÓN IP ORIGEN | | |
| DIRECCIÓN IP DESTINO | | |
| ZERO | PROTOCOLO | LONGITUD TCP |

El campo **puntero urgente** sólo es válido cuando el flag **URG** está activado. Este puntero representa un desplazamiento positivo que debe añadirse al campo **número de secuencia** del segmento para calcular el número de secuencia del último octeto de los datos urgentes. El modo urgente TCP es una forma de que el emisor transmita datos de emergencia ( control ) al otro extremo, de modo que el receptor notifique su llegada a la aplicación tan pronto como sea posible.

La cabecera TCP puede contener las siguientes **opciones** :

- Fin de lista de opciones
- Sin operación
- Tamaño máximo del segmento
- Factor de escala del ventana
- Timestamp

| | | | | |
|---|---|---|---|---|
| FIN DE LISTA DE OPCIONES | CLASE=0 | | | |
| | 1 byte | | | |
| SIN OPERACIÓN | CLASE=1 | | | |
| | 1 byte | | | |
| TAMAÑO MÁXIMO DE SEGMENTO | CLASE=2 | LONG = 4 | TMS | |
| | 1 byte | 1 byte | 2 byte | |
| FACTOR DE ESCALA DE VENTANA | CLASE=3 | LONG = 3 | ESCALA | |
| | 1 byte | 1 byte | 1 byte | |
| TIMESTAMP | CLASE=8 | LONG=10 | TIMESTAMP | TIMESTAMP DE RESPUESTA |
| | 1 byte | 1 byte | 4 byte | 4 byte |

Cada opción comienza con un octeto que especifica su **tip**o. Las opciones con tipo 0 ó 1 ocupan un solo octeto, mientras que el resto de las opciones tienen un segundo octeto que indican su **longitud total,** incluyendo los dos octetos de tipo y longitud.

- La opción de **fin de lista de opciones** indica el final del campo opciones y el

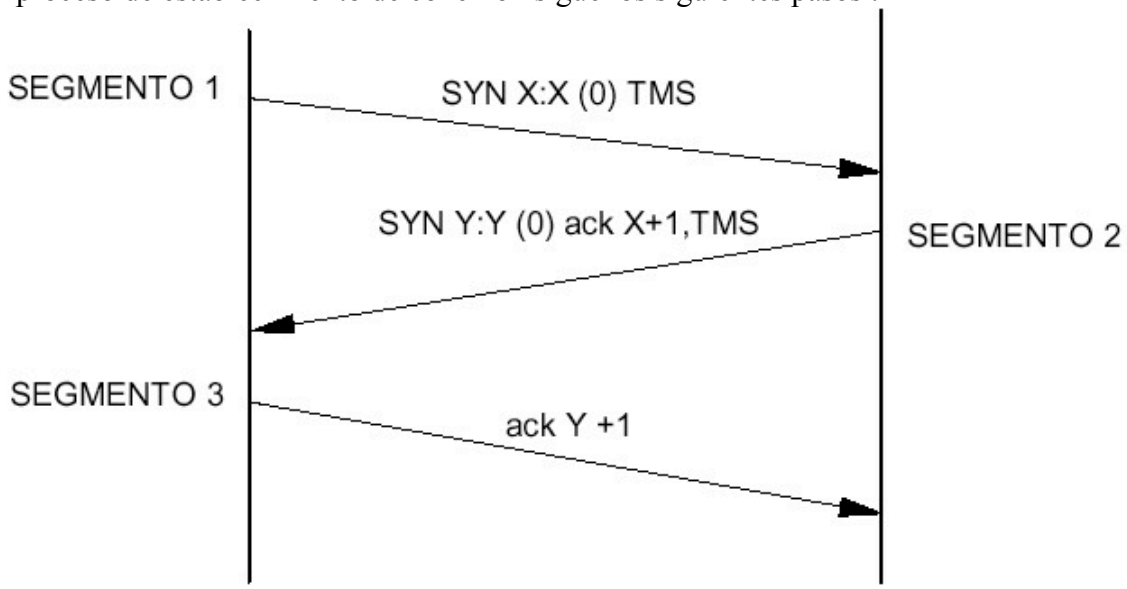
inicio del campo de datos.

- La opción de **sin operación** permite al emisor rellenar campos para hacer que la cabecera sea múltiplo de cuatro octetos.
- Cada extremo de una conexión especifica en el primero de los segmentos intercambiados el **tamaño máximo del segmento** que el desea recibir.
- La opción de **factor de escala de ventana** permite extender el tamaño de la ventana más allá del límite de 65535 octetos, fijando mediante esta opción un factor de escala al valor indicado en el campo tamaño de ventana.
- La opción de **timestamp** permite intercambiar entre ambos extremos los valores del reloj del sistema, si bien no todas las implementaciones hacen uso de esta opción.

Existen otras opciones anteriores que han quedado obsoletas en las últimas revisiones de TCP y algunas recientemente introducidas que todavía no han sido oficialmente aceptadas.

## ESTABLECIMIENTO DE CONEXIONES TCP

TCP es un protocolo orientado a la conexión: antes de que cualquiera de los dos extremos pueda enviar datos al otro debe establecerse una conexión previa entre ellos. El proceso de establecimiento de conexión sigue los siguientes pasos :




El extremo que inicia la conexión ( normalmente el cliente ) envía un primer segmento con el flag **SYN** activado, especificando el **número de puerto** del servidor al que el cliente desea conectarse y el **número inicial de secuenci**a.

El servidor responde con su propio segmento **SYN** conteniendo su **número inicial de secuenci**a.

Asimismo, con este segmento el servidor reconoce la recepción del segmento SYN del cliente enviando en el campo **número de reconocimiento** el valor NSI + 1 ( correspondiente al primer octeto que espera recibir; como puede verse, el segmento SYN consume un número de secuencia.

El cliente debe reconocer este segmento SYN procedente del servidor enviando un tercer segmento 3 en cuyo campo número de reconocimiento incluirá el número del primer octeto que espera recibir. El intercambio de estos 3 segmentos completa el establecimiento de la conexión. El extremo que envía el primer SYN se dice que efectúa una **apertura activ**a, mientras que el otro extremo efectúa una **apertura pasiv**a.

Cuando cada uno de los extremos envía su segmento SYN para establecer la conexión, elige un número de secuencia inicial para dicha conexión. Este valor de NSI debe cambiar en el tiempo de forma que cada conexión emplee un valor diferente. El valor de NSI debe ser contemplado como un contador de 32 bits que se incrementa en uno cada 4 microsegundos; el propósito de estos números de secuencia es evitar que paquetes que hayan quedado retrasados en la red puedan ser mal interpretados como partes de una conexión existente.

Si después de enviar un segmento SYN no se recibe una contestación después de 6 segundos, se envía un segundo segmento SYN. En caso de no recibir tampoco una contestación se enviará un tercer segmento SYN 24 segundos después. Si finalmente tampoco se recibe ningún reconocimiento se descartará la posibilidad de establecer la conexión.

El número de secuencia inicial elegido por cada host es un valor aleatorio. En los RFC's que describen TCP se recomienda utilizar para esto un reloj de sistema que cuente el tiempo en unidades de 4 μs (con un número de secuencia de 32 bits esto supone que el contador se da la vuelta cada 4 horas 46 minutos). De esta forma se reduce la probabilidad de que si uno de los dos hosts cae y rearranca pueda coincidir el número de secuencia nuevo con el viejo, y tomar como válidos segmentos retrasados, o continuar el otro TCP dialogando con el nuevo proceso como si fuera el viejo. Para aumentar la seguridad el RFC recomienda que el software de TCP no esté operativo antes de 2 minutos después de la caída (2 minutos es el tiempo de vida o TTL máximo recomendado de un paquete en la red).

Para comprender hasta que punto es importante la no coincidencia de números de secuencia entre sesiones debemos recordar que cada conexión TCP se identifica por la combinación de los cuatro valores (dirección IP origen, puerto origen, dirección IP destino, puerto destino); supongamos que los usuarios X e Y mantienen ambos una conexión telnet desde la máquina 167.172.23.43 a la máquina 144.38.76.3; en un primer momento sus conexiones podrían ser por ejemplo:

Usuario X: (167.172.23.43, 1024) con (144.38.76.3, 23)
Usuario Y: (167.172.23.43, 1025) con (144.38.76.3, 23)

(1024 y 1025 son puertos utilizados por los procesos cliente telnet en 167.172.23.43 y 23 es el puerto utilizado por el proceso servidor telnet en 144.38.76).

Supongamos ahora que el sistema cliente (host 167.172.23.43) cae y se levanta a continuación, y que los dos usuarios reanudan sus respectivas sesiones telnet (que han quedado medio abiertas en el servidor) pero ahora se conecta Y antes que X; dado que los puertos se asignan por orden de llegada ahora se asignarían en orden inverso:

Usuario X: (167.172.23.43, 1025) con (144.38.76.3, 23)
Usuario Y: (167.172.23.43, 1024) con (144.38.76.3, 23)

En condiciones normales los clientes telnet intentarán abrir nuevas conexiones, con lo que el servidor cerrará las antiguas. Pero si de alguna forma los clientes telnet entraran en las conexiones viejas del servidor sin cerrarlas cada usuario entraría directamente en la sesión del otro sin necesidad de identificarse, lo cual evidentemente no es aceptable. La utilización de números de secuencia grandes y aleatorios suministra un cierto nivel de seguridad ante estas situaciones.

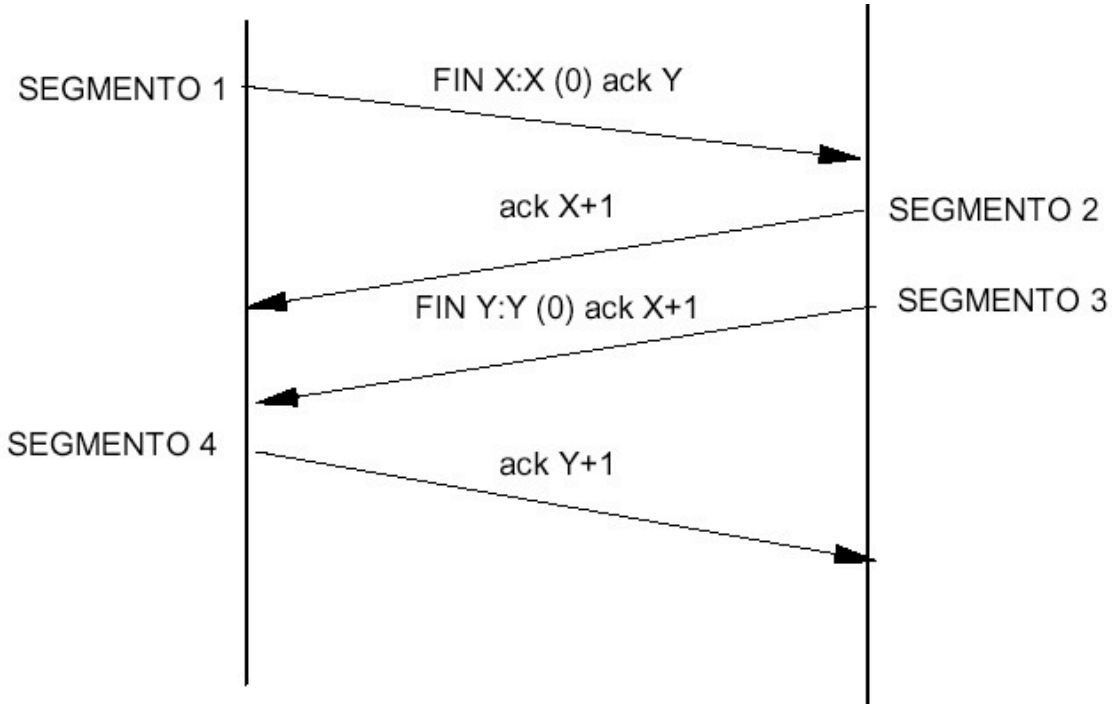
## LIBERACION DE UNA CONEXIÓN TCP

Así como son necesarios 3 segmentos para establecer una conexión, son necesarios 4 para terminar la misma. Las conexiones TCP son full dúplex, de forma que cada sentido debe ser cerrado de modo independiente.

Cualquiera de los extremos puede enviar un segmento FIN cuando ha terminado de enviar datos.

Cuando TCP recibe un segmento FIN, notifica a la aplicación del otro extremo su recepción mediante un ACK en un segmento, indicando que ha terminado el flujo de datos en ese sentido. El envío de un segmento FIN es normalmente el resultado de una petición de cierre por parte de la aplicación. La recepción de un segmento FIN sólo significa que no habrá más datos circulando en ese sentido, sin embargo, TCP puede seguir enviando datos después de haber recibido un FIN. Si bien las aplicaciones pueden hacer uso de esta ventaja de cierre parcial de una conexión, en la práctica, muy pocas aplicaciones TCP hacen uso de ella.

El extremo que efectúa el cierre (el que envía el primer segmento FIN) se dice que efectúa un **cierre activo** mientras que el otro extremo efectúa un **cierre pasiv**o. Normalmente uno de los extremos efectúa el cierre activo mientras que el segundo efectúa el cierre pasivo pero veremos más adelante que ambos extremos pueden efectuar un cierre activo a la vez.



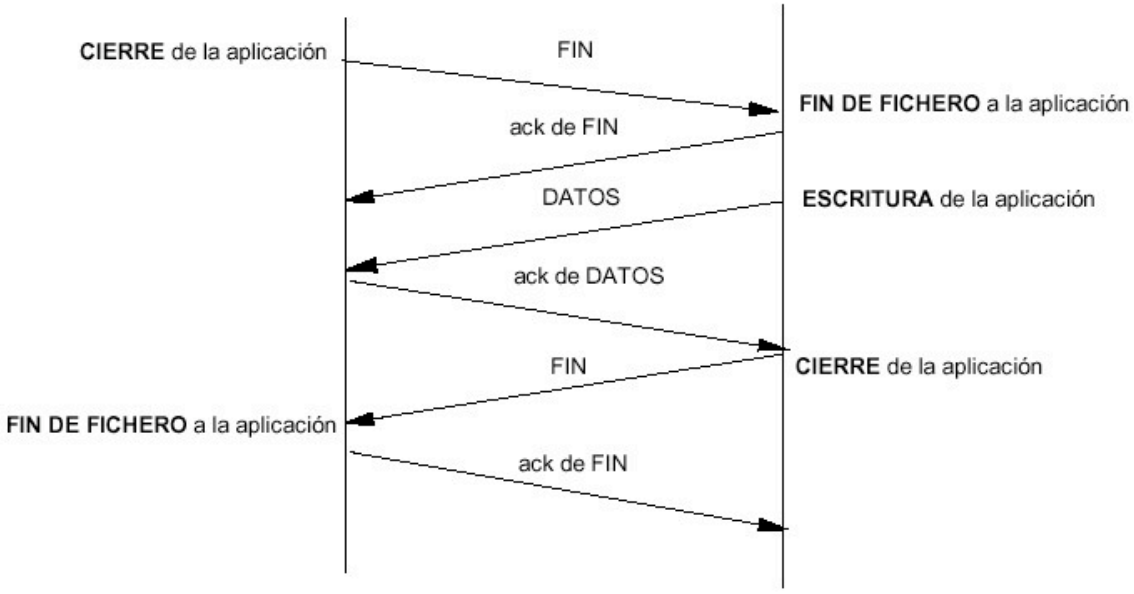

En la figura anterior están representados los segmentos intercambiados entre los extremos de una conexión para liberar la misma. Uno de los dos extremos envía un primer segmento FIN a petición de su aplicación, indicando que ha finalizado la transmisión de datos. Cuando el otro extremo recibe el segmento FIN devuelve un segmento reconociendo (ACK) el número de secuencia recibido mas uno (un FIN consume un número de secuencia, al igual que un SYN). Este segundo extremo cierra su conexión haciendo que TCP envíe un segmento FIN que debe ser reconocido incrementando en uno el número de secuencia del campo reconocimiento del último segmento.

Las conexiones se inician normalmente por el cliente, haciendo que el segmento SYN vaya desde el cliente al servidor, sin embargo cualquiera de los dos extremos puede efectuar un cierre activo de la conexión. Generalmente, es el cliente el que determina cuando debería terminarse la conexión dado que el proceso cliente está controlado generalmente por un uso interactivo que es quien toma la decisión de terminar la conexión.

## SEMICIERRE DE UNA CONEXIÓN TCP

TCP proporciona la posibilidad de que uno de los extremos de una conexión cierre la misma en uno de los sentidos mientras está todavía recibiendo datos procedentes del otro extremo; esto es lo que se denomina **semicierr**e. Pocas son, sin embargo, las aplicaciones que hacen uso de esta capacidad.

Para poder cerrar uno de los sentidos de la comunicación es necesario que la aplicación notifique que ha terminado de enviar datos para que TCP envíe un segmento FIN al otro extremo, sin embargo, es posible continuar recibiendo datos hasta recibir igualmente un segmento FIN .




La figura indica una situación típica de un **semicierr**e. Uno cualquiera de los extremos inicia el semicierre, siendo los dos primeros segmentos los mismos intercambiados en un cierre normal : un FIN seguido de un ACK del FIN recibido. Sin embargo, a diferencia del cierre normal, el extremo que ha recibido el semicierre todavía puede seguir enviando datos. Si bien en la figura sólo se muestra un nuevo segmento de datos seguido de un reconocimiento, el número de segmentos de datos que pueden enviarse es indefinido. Cuando el extremo que recibió el semicierre ha terminado de enviar datos, cierra su extremo de la conexión, haciendo que se envíe un FIN, cuando este segundo FIN es reconocido, la conexión se encuentra cerrada por completo.

## TAMAÑO MAXIMO DEL SEGMENTO

El tamaño máximo del segmento es la cantidad máxima de datos que TCP puede enviar al otro extremo. Cuando se establece una conexión, cada uno de los extremos puede notificar un valor para el **tamaño máximo del segmento** ( TMS ). Este valor no es negociado sino que cada uno de los extremos notifica cual es el tamaño que espera recibir. Si un extremo no recibe una opción TMS del otro supone un valor por defecto de 536 bits.

En general será preferible utilizar un valor de TMS elevado evitando en todo caso que se produzca fragmentación. Un valor más grande en el tamaño del segmento permite que se envíen más datos en cada segmento, minimizando la sobrecarga que suponen las cabeceras TCP e IP. Cuando TCP envía un segmento SYN, bien porque una aplicación local desea iniciar una conexión, o como contestación a una petición de conexión procedente de otro nodo, se puede enviar un valor de TMS como máximo igual al tamaño del MTU del Interfaz a emplear menos el tamaño de las cabeceras IP y TCP.

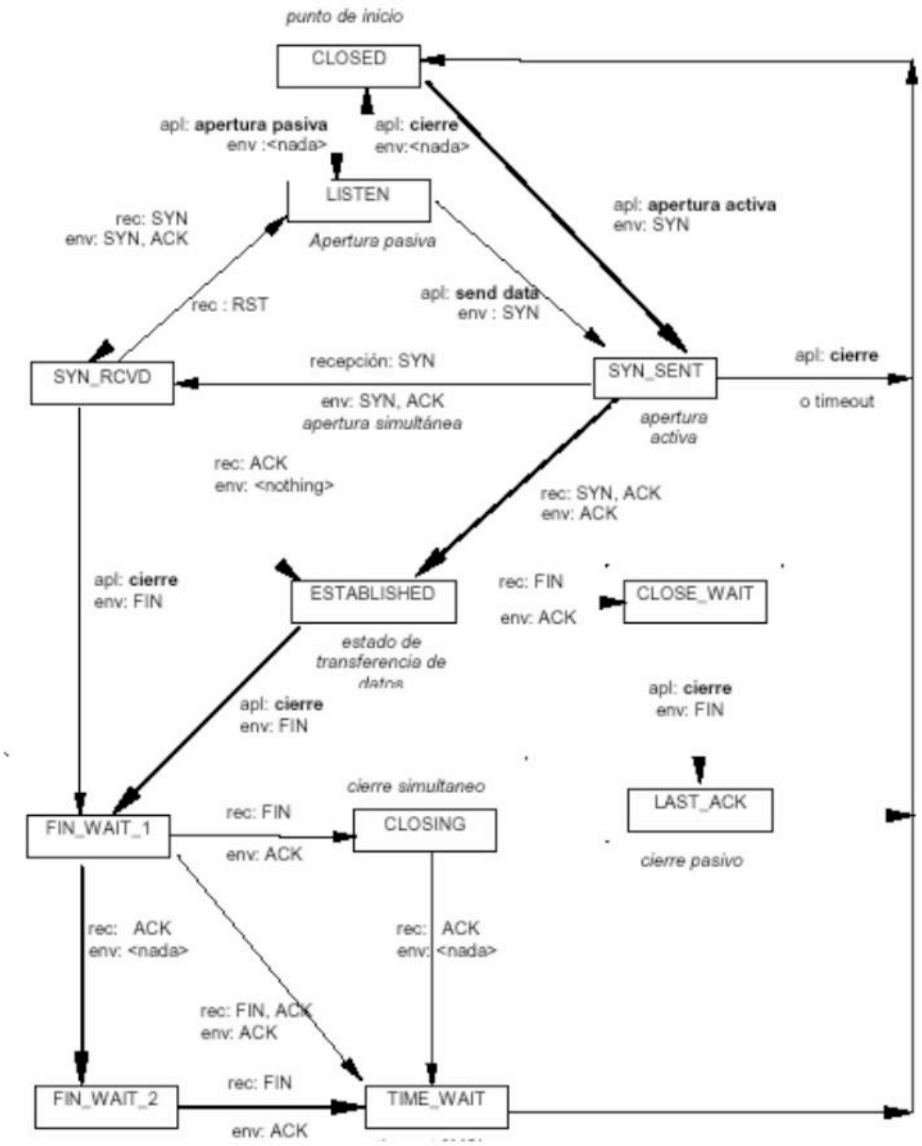
Cuando la dirección IP de destino es “no local”, el valor de TMS se fija por defecto a 536 octetos. Si bien es fácil decir que un destino cuya dirección IP tiene el mismo identificador de red y el mismo identificador de subred que el nuestro es local, y que un destino cuya dirección IP tiene un identificador de red totalmente distinto del nuestro es no local, un destino con el mismo identificador de red pero distinto identificador de subred puede ser bien local o no local. La mayoría de las implementaciones

proporcionan una opción de configuración que permite al administrador del sistema especificar si diferentes subredes son locales o no locales.

TMS fija el límite del tamaño de los datagramas que el otro extremo puede enviar. Cuando se combina con el hecho de que un nodo también puede limitar el tamaño de los datagramas que envía, nos es posible evitar la fragmentación cuando el nodo está conectado a una red con una MTU pequeña.

## DIAGRAMA DE TRANSICIÓN DE ESTADOS DE TCP

Hemos descrito numerosas reglas relacionadas con el inicio y la terminación da la conexión TCP, todas estas reglas pueden ser resumidas en un diagrama de transición de estados como es el que se muestra en la figura.




En el diagrama de transición de estados se han marcado dos subconjuntos habituales de

este diagrama de transición de estados que corresponden a las transiciones de un cliente y de un servidor respectivamente.

Las dos transiciones que conducen al estado ESTABLISHED corresponden a la apertura de una conexión, y las dos transiciones que siguen al estado ESTABLISHED corresponden a la terminación de una conexión. El estado ESTABLISHED corresponde a aquel en el cual se produce la transferencia de datos en ambos sentidos entre los dos extremos.

Hemos recogido en dos bloques cerrados denominados *cierre activo* y *cierre pasivo* los cuatro y dos estados respectivamente que corresponden a cada uno de estos dos cierres.

Los nombres que hemos dado a los once estados de la figura son los que proporciona el comando Unix NETSTAT que ha su vez son idénticos a los nombres empleados originalmente en la especificación de TCP.

El estado CLOSED no es realmente un estado, sino un punto de arranque imaginario y un punto final del diagrama.

La transición de estado desde LISTEN a SYN_SENT es posible, pero no está soportada en las implementaciones derivadas del Unix de Berkley.

La transición desde SYN_RCVD hasta LISTEN sólo es válida si se accedió al estado SYN_RCVD a partir del estado LISTEN y no desde el estado SYN_SENT ( una apertura simultánea ); esto significa que si efectuamos una apertura pasiva ( pasamos al estado LISTEN ), recibimos un SYN, enviamos un SYN con un ACK ( entramos en el estado SYN_RCVD ), y recibimos un RST en lugar de un ACK, este extremo entrará en el estado LISTEN y esperará a que llegue otra petición de conexión.

## ESTADO TIME_WAIT

Cada implementación elige un valor para el **tiempo máximo de vida del segmento** o **MSL** ( maximum segment life ); este valor es el tiempo máximo que un segmento puede permanecer en la red antes de ser descartado. Este tiempo está limitado, puesto que los segmentos TCP se transmiten en datagramas IP, y los datagramas IP tienen un campo TTL que limita su tiempo de vida.

TCP especifica para MSL un valor de dos minutos, si bien es frecuente encontrar valores de treinta segundos, un minuto o dos minutos en distintas implementaciones.

El uso que se hace del valor de MSL es el siguiente: cuando TCP efectúa un cierre activo, y envía el último ACK, la conexión debe permanecer en el estado TIME_WAIT al menos un tiempo igual al doble del valor de MSL; es por esto que este estado se conoce también como TIME_WAIT_2MSL.

La existencia de este estado permite a TCP reenviar el ACK final en caso de que se haya perdido ( en cuyo caso en el otro extremo habrá expirado el temporizador y se retransmitirá su FIN ).

El efecto más importante del estado TIME_WAIT es que mientras la conexión TCP se encuentra en dicho estado, el par de sockets que define la conexión ( dirección IP y número de puerto del cliente y dirección IP y número de puerto del servidor) no puede ser reutilizado; esa conexión sólo puede ser reutilizada cuando se abandone el estado TIME_WAIT.

Sin embargo, la mayoría de las implementaciones imponen una restricción más exigente: un número de puerto local no puede reutilizarse mientras que dicho puerto corresponda a un socket que está en el estado TIME_WAIT.

Cualquier segmento retrasado que llegue correspondiente a una conexión que todavía se encuentra en el estado TIME_WAIT es descartado. Dado que la conexión definida por el socket que se encuentra en estado TIME_WAIT no puede ser reutilizada durante este periodo de tiempo, cuando establecemos una conexión válida existe la certeza de que los segmentos retrasados correspondientes a conexiones anteriores no pueden ser malinterpretados como pertenecientes a esta nueva conexión.

Como se puede ver en el diagrama de transición de estados, es normal que el cliente sea quien efectúe el cierre activo y entre en el estado TIME_WAIT. El servidor generalmente efectúa el cierre pasivo y no atraviesa dicho estado de espera; la implicación de esto es que cuando se termina un cliente, y se rearranca inmediatamente, el nuevo cliente no puede utilizar el mismo número de puerto local. Esto no supone ningún problema dado que los clientes generalmente utilizan números de puertos aleatorios sin preocuparse de cuál sea éste. Los servidores, sin embargo, utilizan números de puertos conocidos. Si un servidor cierra la conexión, e inmediatamente después intentamos rearrancarla, el servidor no podrá asignar su número de puerto a su extremo, dado que el número de puerto es todavía parte de una conexión que está en estado TIME_WAIT. Llevará de 1 a 4 minutos poder rearrancar el servidor.

La espera 2MSL proporciona una protección contra los segmentos retrasados correspondientes a conexiones anteriores, sin embargo esto solo funciona si un nodo con conexiones en el estado de espera 2MSL no falla; si es así, y rearranca dentro del tiempo correspondiente al estado TIME_WAIT e inmediatamente establece una nueva conexión utilizando las mismas direcciones IP y números de puerto correspondientes a una conexión anterior, segmentos que pudieran permanecer retrasados correspondientes a conexiones anteriores antes del fallo podrían ser malinterpretadas como pertenecientes a las nuevas conexiones creadas después del rearranque, y esto puede suceder independientemente de cómo se elija el número inicial de secuencia después del rearranque.

Para eliminar esta posibilidad, TCP no debería crear ninguna conexión durante un número de segundos igual a MSL después de rearrancar.

## ESTADO FIN_WAIT_2

En el estado FIN_WAIT_2 uno de los extremos ha enviado un FIN y el otro extremo lo ha reconocido.

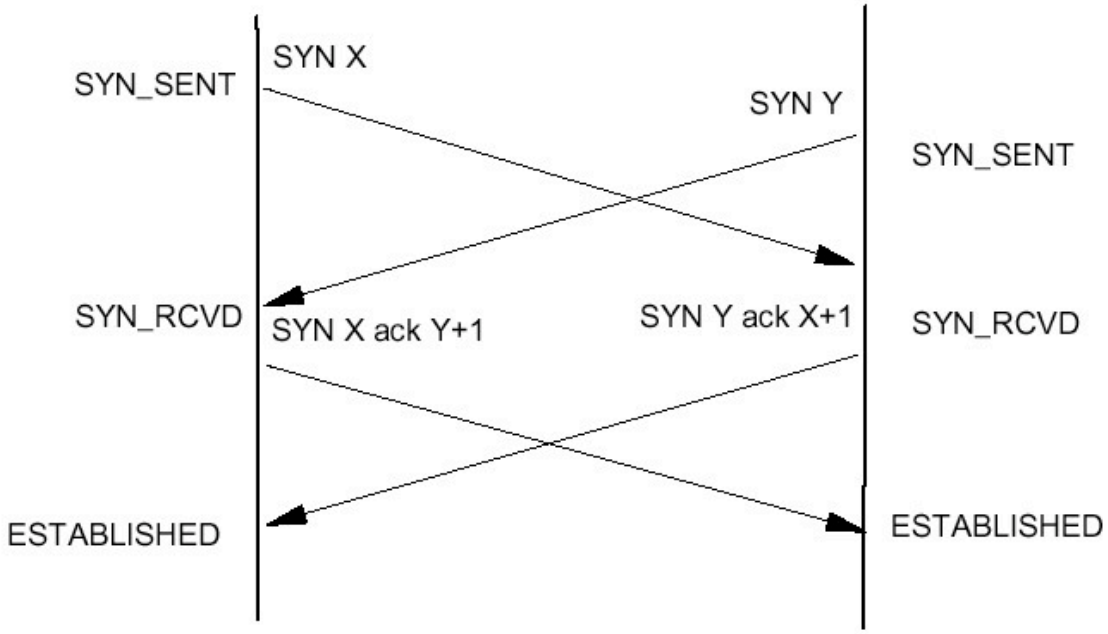
Salvo en el caso de un semicierre, TCP está esperando que la aplicación del otro

extremo reconozca que ha recibido una notificación de final de fichero y cierre su extremo de la conexión, enviando un FIN. Sólo cuando el proceso en el otro extremo efectúe ese cierre nuestro extremo pasará del estado FIN_WAIT_2 al estado TIME_WAIT.

Esto significa que un extremo de la conexión puede permanecer en este estado indefinidamente: el otro extremo está todavía en el estado esperando cierre, y puede permanecer ahí definidamente hasta que la aplicación decida efectuar su cierre.

## APERTURA SIMULTÁNEA

Es posible, si bien improbable, que dos aplicaciones efectúen una apertura activa a la vez, transmitiendo un SYN. Esto supone que cada uno de los extremos tenga un número de puerto local conocido por el otro extremo, y es denominado apertura simultánea. TCP ha sido diseñado para poder gestionar aperturas simultáneas creando una sola conexión en lugar de dos como haría un protocolo OSI.



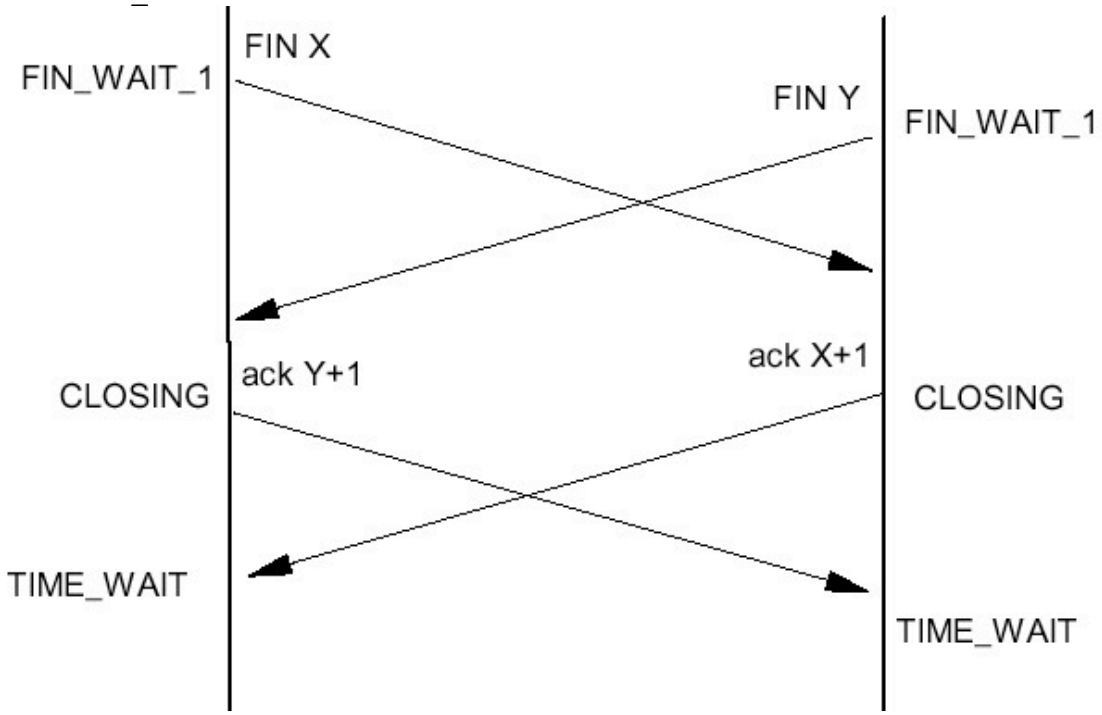

Cuando se produce una apertura simultánea, ambos extremos envían un segmento SYN al mismo tiempo, entrando en el estado SYN_SENT. Cuando cada uno de los extremos recibe un SYN, cambian el estado SYN_RCVD, reenviando cada uno de ellos el SYN y reconociendo el SYN recibido. Cuando cada extremo recibe el segmento SYN y ACK, cambian su estado al estado ESTABLISHED.

Como puede verse, una apertura simultánea necesita del intercambio de cuatro segmentos en lugar de los tres que habitualmente se intercambian en la negociación de establecimiento de conexión.

## CIERRE SIMULTÁNEO

Anteriormente hemos dicho que es habitualmente el cliente quien efectúa el cierre activo, originando el envío del primer FIN. Sin embargo, también posible que ambos extremos efectúen un cierre activo a la vez, lo que en el protocolo TCP se conoce como **cierre simultáne**o.

Como puede verse en la figura ambos extremos van desde el estado ESTABLISHED al estado TIME_WAIT_1 cuando la aplicación efectúe el cierre. Esto hace que se envíen dos FIN, que cuando sean recibidos harán que cambien los estados a CLOSING, y cada uno de los extremos enviará su segmento ACK, que al ser recibidos conducirán al estado TIME_WAIT.




## SEGMENTOS DE REINICIO ( RESET )

Hemos mencionado anteriormente la existencia de un flag en la cabecera TCP denominado RST (reinicio). En general, TCP envía un reinicio siempre que llega un segmento que no parece ser correcto en el marco de la conexión referenciada.

Un caso general para enviar un reinicio es aquel que se produce cuando llega una petición de conexión y no hay ningún proceso asignado al número de puerto de destino. En un caso como éste, UDP enviaría un mensaje de puerto no alcanzable a la vez que descartaría el datagrama, en su lugar, TCP envía un segmento de reinicio ( RST ).

El modo normal de terminar una conexión es el envío de un segmento FIN, esto es lo que generalmente se conoce como una liberación ordenada, sin embargo, es posible abortar una conexión enviando un segmento de reinicio ( RST ) en lugar de un segmento de FIN, lo que se conoce en ocasiones como una liberación desordenada.

Abortar una conexión tiene dos consecuencias para la aplicación : cualquier dato encolado es descartado y el reinicio se envía de forma inmediata, y el receptor de un RST sabe que el otro extremo abortó la conexión en lugar de efectuar un cierre normal.

Se dice que una conexión TCP está semiabierta si uno de los extremos ha cerrado o abortado la conexión sin conocimiento del otro extremo, lo que sucede solamente en caso de fallo en uno de los dos nodos. Dado que no hay ningún intento de transmitir

datos a través de una conexión semiabierta, el extremo que todavía está en funcionamiento no detectará que el otro extremo ha fallado.

Otra causa común de la existencia de conexiones semiabiertas es la costumbre de desconectar los clientes en lugar de terminar la aplicación y después apagar el nodo. Si no se estaba transfiriendo datos cuando el cliente se desconectó el servidor nunca sabrá que el cliente desapareció. Cuando el usuario rearranque su aplicación cliente, se arrancará una nueva ocurrencia del servidor en el nodo servidor, lo que puede conducir a la existencia de múltiples conexiones semiabiertas TCP. ( veremos más adelante un procedimiento para que un extremo de una conexión semiabierta descubra que el otro extremo ha desaparecido utilizando una opción de TCP ).

Sin embargo, cuando se produce un fallo en un servidor y rearranca sin haber cerrado las conexiones que mantenía con los clientes, cuando estos envíen nuevos segmentos correspondientes a una conexión de la que el servidor TCP no tiene conocimiento, éste devolverá segmentos de reinicio ( RST ) que provocarán el cierre de las conexiones en el lado del cliente.

## FLUJO DE DATOS INTERACTIVOS TCP

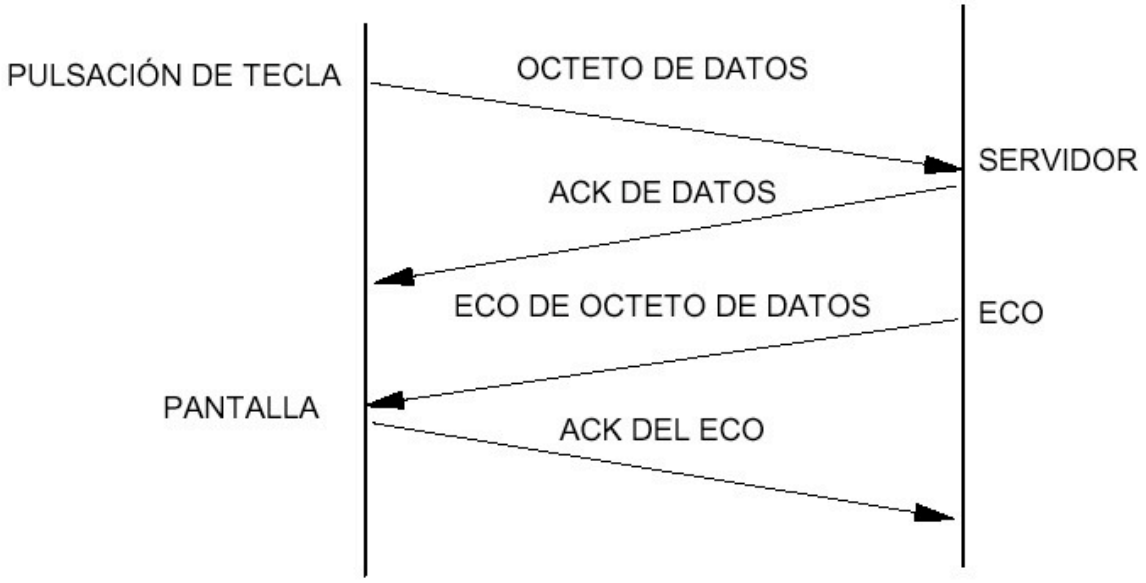
Las aplicaciones que hacen uso de TCP generan los datos que envían a través de una conexión bien de forma masiva ( FTP, News, correo electrónico ) o bien carácter a carácter ( Telnet y Rlogin ), estas últimas aplicaciones se conocen como interactivas.

La proporción entre los segmentos procedentes de ambos tipos de aplicaciones es la misma, pero sin embargo, desde el punto de vista de octetos transmitidos la relación es del 90% de datos masivos y del 10% de datos interactivos, la razón es que los segmentos de datos interactivos son muy pequeños ( uno o varios octetos ) mientras que los otros segmentos suelen ir llenos ( normalmente 512 octetos de datos de usuario ).

Obviamente TCP gestiona ambos tipos de datos, sin embargo, emplea diferentes algoritmos para cada uno de ellos.

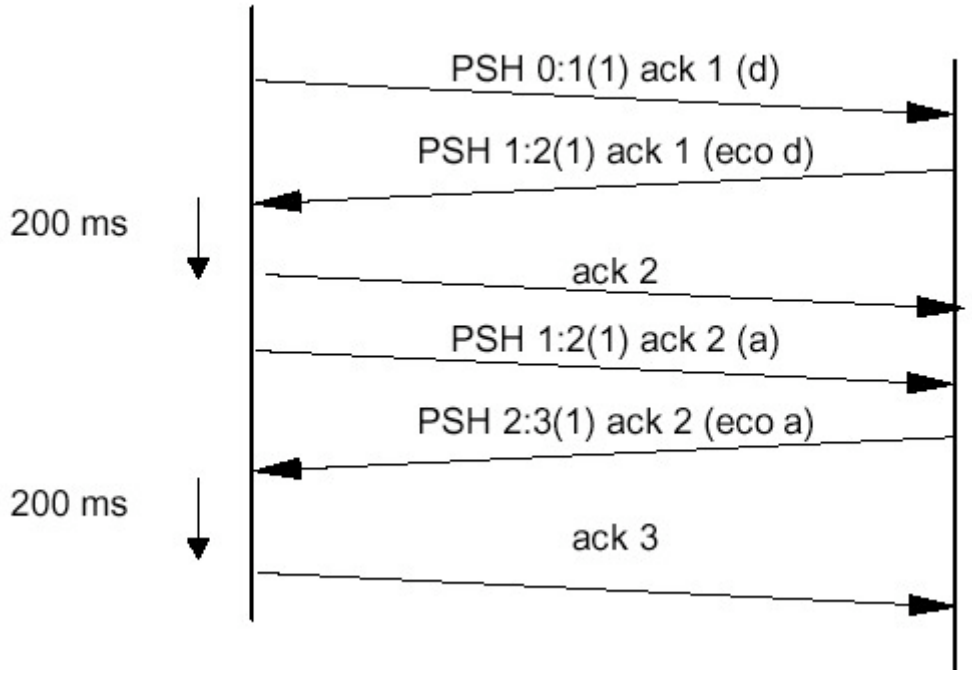
## ENTRADA INTERACTIVA

Si analizamos el flujo de datos generado por una aplicación como Rlogin, veremos que cada pulsación del teclado genera un segmento de datos conteniendo un octeto ; si además el sistema remoto hace eco de los caracteres recibidos, podrían generarse cuatro segmentos por cada pulsación del teclado tal y como vemos en la figura.




## RECONOCIMIENTOS DIFERIDOS

En la figura siguiente puede apreciarse el funcionamiento habitual de TCP cuando trata datos interactivos. TCP no envía un ACK en el mismo instante en que recibe datos, sino que lo difiere esperando tener datos que enviar en el mismo sentido ( "piggyback" ) .




La mayoría de las implementaciones utilizan un retraso de 200 milisegundos, es decir, TCP diferirá todo ACK durante 200 milisegundos para ver si tiene datos que enviar junto con él.

## ALGORITMO DE NAGLE

El envío separado que de cada una da las teclas pulsadas en una aplicación interactiva desde el cliente hasta el servidor genera datagramas IP de 41 octetos : 20 octetos de

cabecera IP, 20 de cabecera TCP y 1 octeto de datos. Estos paquetes pequeños ( "tinygrans" ) no constituyen un problema en las redes de área local, que generalmente no están congestionadas, pero pueden originar una congestión en las redes de área extensa; es por ello que habitualmente se utiliza para evitarlo una solución basada en el **algoritmo de nagl**e.

Según este algoritmo, una conexión TCP no puede tener más de un segmento pequeño pendiente de ser reconocido, por tanto, no podrá enviar nuevos segmentos pequeños hasta no haber recibido el reconocimiento del segmento pendiente, y deberá almacenar los datos introducidos hasta que llegue el reconocimiento y enviarlos en un único segmento. Cuanto más deprisa lleguen los reconocimientos, más deprisa serán enviados los datos, pero en un red extensa que sea muy lenta, donde sería deseable reducir el número de tinygrans, se enviarán menos segmentos.

La idea es muy simple, cuando el tráfico de la aplicación llega al TCP en bytes sueltos se envía el primero y se retienen los demás hasta recibir el ACK correspondiente al byte enviado; una vez recibido el ACK se envía un segmento con los bytes que hubiera pendientes y se empieza a acumular de nuevo hasta recibir el siguiente ACK. También se envía un segmento si el número de caracteres acumulados en el buffer es igual a la mitad de la ventana, o al máximo tamaño del segmento.

En cierto modo el algoritmo de Nagle sustituye el protocolo de ventana deslizante por un mecanismo de parada y espera.

Hay ocasiones en las que el algoritmo de nagle debe ser desactivado, el ejemplo más clásico es el de un servidor XWINDOW donde los datagramas que corresponden a movimientos del ratón deben ser enviados sin retrasos para proporcionar un movimiento en tiempo real correspondiente a las acciones del usuario.

## FLUJO MASIVO DE DATOS TCP

El mecanismo de control de flujo que emplea TCP es el de **ventana deslizante** pudiendo variar el tamaño de la ventana en cada momento en función de la memoria que tenga disponible el receptor.

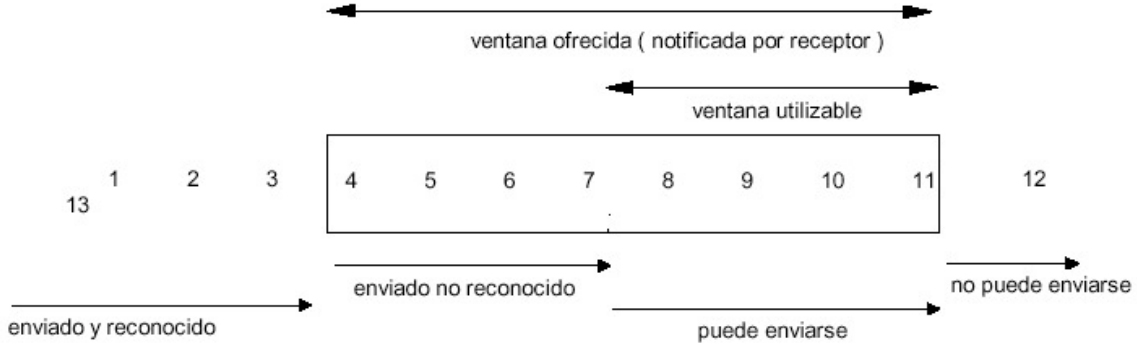
En la figura hemos representado el protocolo de ventana deslizante empleado por TCP La **ventana notificada por el receptor** se denomina habitualmente **ventana ofrecid**a. Comienza en el primer octeto no reconocido y el emisor, a partir de su longitud, calcula la **ventana utilizable** que le indica cuántos datos puede enviar de forma inmediata.

A lo largo del tiempo esta ventana deslizante se desplaza hacia la derecha a medida que el receptor reconoce datos. El movimiento relativo de los dos extremos de la ventana hace aumentar o disminuir el tamaño de la misma ; se utilizan tres términos para describir el movimiento de los extremos izquierdo y derecho de la ventana :

- La ventana se **cierra** a medida que le extremo izquierdo avanza hacia la derecha, lo que se produce cuando se envían reconocimientos.
- La ventana se **abre** cuando el extremo derecho se mueve hacia la derecha permitiendo que sean enviados más datos; esto se produce cuando el proceso

receptor del otro extremo lee datos reconocidos liberando espacio en su buffer de recepción TCP.

- La ventana se **reduce** cuando la distancia entre los extremos de la ventana disminuye. Es posible que el extremo derecho se mueve hacia la izquierda, si bien esta situación no es recomendable; el tamaño de la ventana puede disminuir manteniendo el extremo derecho de la ventana en su sitio y avanzando el extremo izquierdo con reconocimientos.




El tamaño de la ventana ofrecida es controlado por el proceso de recepción, y puede afectar notablemente al rendimiento TCP. Son habituales tamaños del buffer de emisión y recepción de 2048, 4096, 8192 o 16384 octetos. El tamaño del buffer de recepción es el tamaño máximo de la ventana que puede notificarse para dicha conexión, y algunas aplicaciones pueden alterar su tamaño para incrementar el rendimiento.

El aumento del tamaño del Buffer de 4096 a 16384 supone un aumento en el rendimiento de un 40 %.

Sin embargo, nos encontramos con un caso especial a tratar. Imaginemos una aplicación que genera datos con rapidez y que los envía a otra, la cual los recoge del buffer de TCP a razón de un byte cada vez. La situación que se daría sería la siguiente:

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

El bucle se repite hasta terminar la sesión. Se están generando como antes segmentos con un byte de información útil, pero esta vez el causante es el receptor. Este comportamiento se conoce como síndrome de la ventana tonta.

La solución fue propuesta por Clark en la RFC 813; el TCP receptor no debe notificar el cambio de ventana al emisor entretanto no tenga una cantidad razonable de espacio libre en su buffer; por razonable se entiende el espacio suficiente para aceptar un segmento de la longitud máxima admitida en esa conexión, o la mitad del espacio total de buffer, lo que ocurra primero.

## FLAG PUSH

Este flag es una notificación del emisor al receptor para que pase todos los datos que tenga al proceso receptor: el contenido del segmento que contiene el flag push activado y todos los datos que TCP había recibido anteriormente para el proceso receptor.

En la especificación original de TCP se supuso que el interfaz de programación permitiría al proceso emisor informar a TCP cuándo debería activar este flag, de este modo, un cliente podría notificar que no desea que los datos permanezcan almacenados en el buffer TCP esperando a datos adicionales antes de ser enviados. Cuando un servidor TCP recibe un segmento con este el flag push activado, sabe que debe pasar los datos al proceso servidor y no esperar a ver si llegan nuevos datos.

La mayoría de los interfaces de programación, sin embargo, no proporcionan un medio para que la aplicación notifique a TCP la activación del flag push; una buena implementación de TCP debe poder determinar por sí misma cuándo debe activar dicho flag.

La mayoría de las implementaciones derivadas de Berkeley activan automáticamente el flag push cuando los datos contenidos en el segmento han dejado vacío el buffer de envío; esto significa que generalmente el flag push está activado para cada escritura de la aplicación, dado que los datos habitualmente se envían cuando se escriben.

Las implementaciones derivadas de Berkeley ignoran los flag push recibidos porque normalmente nunca retrasan el envío de los datos recibidos a la aplicación.

## ARRANQUE LENTO

En todos los ejemplos que hemos visto hasta ahora, el emisor transmite a la red varios segmentos hasta completar el tamaño de ventana notificado por el receptor; si bien esto funciona correctamente cuando los dos nodos se encuentran en la misma red de área local, si hay routers y enlaces más lentos entre el emisor y el receptor pueden llegar a producirse problemas de congestión. Los routers intermedios deberán encolar los paquetes y es posible que alguno de ellos pueda quedarse sin espacio de almacenamiento.

TCP soporta un algoritmo denominado **arranque lento** que opera asegurando que la velocidad a la que se insertan nuevos paquetes en la red es la velocidad a la que se devuelven los reconocimientos por el otro extremo.

El arranque lento añade una nueva ventana al TCP emisor: la **ventana de congestió**n, denominada VENC. Cuando se establece una nueva conexión con un nodo en otra red, la ventana de congestión se inicializa a un segmento (el tamaño de segmento anunciado por el otro extremo). Cada vez que se recibe un ACK, la ventana de congestión se incrementa en un segmento ( VENC se mantiene en octetos, pero el arranque lento siempre lo incrementa en el tamaño del segmento). El emisor puede transmitir hasta alcanzar el valor más pequeño de entre la ventana de congestión y la ventana notificada. La ventana de congestión es un control de flujo impuesto por el emisor, mientras que la ventana notificada es un control de flujo impuesto por el receptor.

El emisor arranca transmitiendo un segmento y esperando su ACK. Cuando dicho ACK se recibe, la ventana de congestión se incrementa de 1 a 2, y será posible enviar dos segmentos. Cuando cada uno de estos dos segmentos sea reconocido la ventana de congestión se incrementará a 4, lo cual proporciona un incremento exponencial en el tamaño de dicha ventana.

En algún punto es posible que se sobre pase la capacidad de la red, y de que un router intermedio comience a descartar paquetes. Esto dice al emisor que la ventana de congestión se ha hecho demasiado grande. Cuando hablemos de los temporizadores TCP y del algoritmo de retransmisión veremos cómo ser gestiona esto y qué ocurre con la ventana de congestión.

## RENDIMIENTO EN LA TRANSMISIÓN DE DATOS "BULK"

Vamos a analizar la relación del tamaño de la ventana, el control de flujo y el arranque lento con el rendimiento de una conexión TCP.

Cada nuevo reconocimiento recibido el emisor amplía la ventana de congestión en un segmento. El emisor devuelve un reconocimiento con cada segmento de datos recibido, y por lo tanto el espaciado de los reconocimientos coincide con el espaciado entre los segmentos enviados por el emisor; la ventana de congestión irá aumentando hasta que los canales de envío y de recepción estén llenos y no puedan contener más datos, independientemente de la ventana de congestión o de la ventana notificada por el receptor. Con cada nuevo segmento que es retirado de la red por el receptor, el emisor pone un nuevo segmento en la red, sin embargo, hay tantos reconocimientos en el camino de vuelta como segmentos en el camino de ida, esto constituye el estado ideal de una conexión.

Una vez visto esto podemos contestar a la cuestión: ¿Cuál debería ser el tamaño de una ventana?

La ventana notificada por el receptor debería ser igual al tamaño en el cual el canal está lleno de datos, puesto que este es el límite de números de segmentos que el emisor puede transmitir.

Podemos calcular esta capacidad como:
Capacidad (bytes = ancho de banda)(bytes/segundo) * Tiempo de retorno (segundos)

Esto se conoce normalmente como producto ancho de banda-retraso. Este valor puede variar dependiendo de la velocidad de la red y del tiempo de retorno entre los dos extremos.
Tanto el ancho de banda como el retraso pueden afectar la capacidad del canal entre el emisor y el receptor. Duplicar el tiempo de retorno duplica la capacidad del canal y de la misma forma, duplicar el ancho de banda también duplica la capacidad del canal.

## MODO URGENTE

TCP proporciona un modo de funcionamiento denominado urgente, que permite a uno de los extremos informar al otro de que se han insertado datos de carácter urgente en el

flujo normal de datos, y corresponde a este extremo receptor decidir qué hacer.

Para notificar la existencia de datos urgentes se utilizan dos campos de la cabecera TCP. El flag URG se activa y el puntero urgente de 16 bytes sirve para indicar un desplazamiento positivo que debe ser añadido al número de secuencia de la cabecera TCP para obtener el número de secuencia del último byte de datos urgentes.

TCP debe informar al proceso receptor de la recepción de un puntero urgente si no hay otro ya pendiente en dicha conexión o si el puntero urgente avanza en el flujo de datos. La aplicación receptora puede leer el flujo de datos y debe ser capaz de decir cuándo se ha encontrado el puntero urgente. Dado que existen datos desde la posición de lectura actual del receptor hasta el puntero urgente, la aplicación se considera en "modo urgente", después de pasar el puntero urgente, la aplicación vuelve a su estado normal.

El propio TCP informa de poco más acerca de los datos urgentes. No hay ningún modo de especificar dónde comienzan los datos urgentes en el flujo de datos. La única información enviada a través de la conexión por TCP es que el modo urgente a comenzado (activando el byte URG de la cabecera TCP) y el puntero indicando el último byte de datos urgentes, todo lo demás queda en manos de la aplicación.

Desafortunadamente muchas implementaciones denominan incorrectamente al modo urgente como modo fuera de banda. Si una aplicación realmente desea un canal fuera de banda, debe establecer una segunda conexión TCP para conseguirlo.

¿Para qué sirve el MODO URGENTE?. Las dos aplicaciones más comunes son TELNET y RLOGIN, cuando usuarios interactivos pulsan la tecla de interrupción, como veremos más adelante al estudiar estos protocolos. Otro ejemplo es FTP, cuando un usuario interactivo cancela la transferencia de fichero como se mostrará en el capítulo correspondiente.

TELNET y RLOGIN utilizan el modo urgente desde el servidor al cliente porque es posible que el cliente detenga la dirección de datos en este sentido (notificando una ventana de tamaño 0). Pero si el proceso del servidor entra en modo urgente, el servidor TCP envía inmediatamente el puntero urgente y el flag URG, incluso aunque no pueda enviar ningún dato. Cuando el cliente TCP recibe esta notificación, lo notifica al proceso cliente, de modo que el cliente puede leer el mensaje del servidor, abrir la ventana y volver a permitir el flujo de datos.

¿Qué ocurre si el emisor entra en modo urgente varias veces antes de que el receptor procese todos los datos hasta el primer puntero urgente?. El puntero urgente simplemente avanza en el flujo de datos, y su posición previa en el receptor se pierde. Sólo hay un puntero urgente en el receptor y su valor es sobrescrito cada vez que llega un nuevo puntero urgente del otro extremo; esto significa que si el contenido del flujo de datos escrito por el emisor cuando entre en modo urgente es importante para el receptor, estos bytes de datos deberían ser marcados especialmente por el emisor. Veremos que TELNET marca todos sus comandos en el flujo de datos precediéndolos con un byte 255.

## TIMEOUT, TCP Y RETRANSMISIÓN

TCP proporciona un nivel de transporte fiable, por una parte permite a cada extremo reconocer los datos procedentes del otro y por otro lado, para contemplar la pérdida tanto de los segmentos como de los reconocimientos emplea un temporizador que cuando expira obliga a retransmitir el segmento.

Un elemento crítico es la decisión de la duración de este temporizador antes de expirar.

TCP emplea cuatro contadores diferentes para cada una de las conexiones:

1. Un temporizador de retransmisión que se emplea cuando se está esperando un reconocimiento del otro extremo.
2. Un temporizador de persistencia que permite transmitir información acerca del tamaño de la ventana aún cuando el otro extremo haya cerrado su ventana de recepción.
3. Un temporizador " " para detectar cuándo el otro extremo de una conexión se ha desconectado o rebotado.
4. Un temporizador 2MSL que mide el tiempo en que una conexión ha estado en el estado tiempo de espera.

### TIMEOUT Y RETRANSMISIÓN

El primer "timeout" se produce 1,5 segundos después de la primera transmisión, a partir de este momento cada nueva retransmisión duplica el valor del "timeout" con un límite superior de 64 segundos. Este esquema se denomina retraso exponencial.

### MEDICIÓN DEL TIEMPO DE RETORNO

Un aspecto fundamental en la temporización de TCP y en la retransmisión es la medición del tiempo de retorno experimentado en una conexión dada. Es de esperar que este valor cambie a lo largo del tiempo con las modificaciones de las rutas y con el cambio en el tráfico de la red, y TCP debería seguir estos cambios y modificar su "timeout".

En primer lugar TCP debe medir el tiempo de retorno entre un byte enviado con un número particular de secuencia y la recepción de un reconocimiento que corresponderá a dicho número de secuencia; hay que tener en cuenta que normalmente no existe una correspondencia uno a uno entre segmentos de datos y reconocimientos.

La especificación TCP original emplea una estimación del tiempo de retorno utilizando un filtro paso bajo:

$$R=\alpha * R + (1-\alpha)M$$

Donde $\alpha$ es un factor de corrección con un valor recomendado de 0,9. Esta aproximación del tiempo de retorno se actualiza cada vez que se efectúa una nueva medida. El 90% de cada nueva estimación proviene de la estimación anterior y el 10% de la nueva medida.
Dado este estimador que cambia a medida que lo hace el valor del tiempo de retorno, el tiempo de retransmisión recomendado por el RFC793 debe ser:

$$RTO= R*\beta$$

Donde $\beta$ es un factor de variancia de retraso con un valor recomendado de 2.

## *EL PROTOCOLO DE DATAGRAMA DE USUARIO: UDP*

Dentro del conjunto de protocolos TCP/IP, el protocolo **UDP** ( **Protocolo de Datagrama de Usuario** ) proporciona al mecanismo primario que los programas de aplicación utilizan para enviar datagramas a otros programas de aplicación. UDP proporciona **puertos de protocolo** que se emplean para distinguir entre múltiples programas que se estén ejecutando en una misma máquina ( demultiplexación ). Por lo tanto, además de los datos enviados, cada mensaje UDP tiene una cabecera que contiene tanto el **número de puerto de destino** como el **número de puerto de origen**, haciendo posible que el protocolo UDP en el destinatario envíe el mensaje al destino correcto y al destinatario enviar una contestación.

UDP utiliza el protocolo IP subyacente para hacer llegar los mensajes desde un sistema a otro, y proporciona el mismo sistema de envío **no fiable** y **no orientado a la conexión** que proporciona IP.

No utiliza reconocimientos para asegurar que los mensajes han llegado, no reordena los mensajes recibidos, y no proporciona control de flujo de información. Así, los mensajes UDP pueden perderse, duplicarse, o llegar fuera de orden, y además, los paquetes pueden llegar más deprisa de lo que el receptor puede procesarlos.

Un programa de aplicación que utiliza UDP acepta la responsabilidad de gestionar el problema de la fiabilidad, incluyendo la pérdida de mensajes, la duplicación de los mismos, el retraso, el envío fuera de orden y la pérdida de conexión.

### IDENTIFICACIÓN DEL DESTINATARIO DE UN DATAGRAMA

La mayoría de los sistemas operativos soportan multiproceso, o lo que es lo mismo, permiten que haya múltiples programas de aplicación ejecutándose simultáneamente, que denominaremos procesos o tareas. Es evidente que el destinatario último de un mensaje será uno de estos procesos, pero la identificación de uno de estos procesos como destino último de un datagrama tiene cierta complejidad.

En primer lugar, los procesos se crean y destruyen de forma dinámica, por ello los emisores de datagramas difícilmente sabrán lo suficiente para identificar un proceso en la máquina de destino. En segundo lugar, los sistemas deben poder reemplazar procesos sin tener que informar a los posibles emisores de datagramas de dichos cambios. Resulta necesario, por todo lo anterior, identificar a los destinos basándose en las funciones que implementan en lugar de en la identificación de los procesos que implementan dicha función.

En lugar de pensar en un proceso como el destinatario último de un datagrama, podemos imaginar que cada sistema contiene un conjunto abstracto de puntos de destino denominados **Puertos de Protocolo**, cada uno de los cuales se identifica mediante un número entero positivo. El sistema operativo proporciona un mecanismo de Interfaces que facilita a los procesos el acceso a los puertos.

La mayoría de los sistemas operativos proporcionan acceso síncrono a los puertos; por ejemplo, si un proceso intenta extraer datos de un puerto antes de que lleguen los propios datos, el sistema operativo detiene (bloquea) el proceso hasta que dichos datos

lleguen. Una vez que los datos llegan, el sistema operativo pasa los datos al proceso y lo rearranca. En general, los puertos se gestionan como un "buffer", de modo que los datos que llegan antes de que el proceso esté listo para aceptarlos se almacenarán y no se perderán; para conseguir esto, el sistema operativo coloca los paquetes que llegan destinados a un puerto de protocolo particular en una cola hasta que el proceso los extrae de ella.

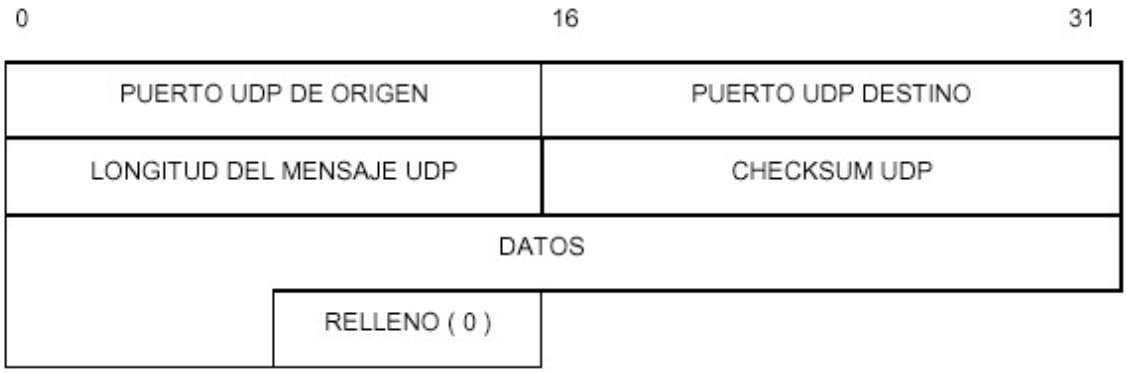
Para comunicarse con un determinado, el emisor necesita saber tanto la dirección IP del sistema de destino como el número de proceso de protocolo del destino dentro de dicho sistema. Cada mensaje contiene tanto el número de puerto de destino en el sistema receptor al cual se envía el mensaje como el número de puerto de origen del sistema emisor al cual deben enviarse las contestaciones; de esta forma, resulta posible para cualquier proceso que reciba un mensaje contestar al emisor.

## FORMATO DE LOS MENSAJES UDP

Cada mensaje UDP, denominado datagrama de usuario, consta de dos partes fundamentales:

- **Cabecera**
- **Área de datos**

En la figura puede verse que la cabecera se divide en cuatro campos de 16 bits que especifican el puerto desde el cual se envía el mensaje, el puerto al cual está destinado el mensaje, la longitud del datagrama de usuario, y un checksum para comprobar que el datagrama llega a su destino sin haber sufrido ninguna alteración.




Los campos **Puerto de Origen** y **Puerto de Destino**, contienen los números de puerto de protocolo de 16 bits que identifican a los procesos emisor y receptor del datagrama de usuario. El campo **Puerto de Origen** es opcional; cuando se utiliza, especifica el puerto al que se debería enviar la contestación; si no se utiliza, debe ser 0.

El campo **Longitud** contiene el número de octetos del datagrama UDP incluyendo la cabecera y el campo de datos de usuario, por lo tanto, el valor mínimo de este campo es 8, la longitud de la cabecera ( es por lo tanto posible enviar datagramas en los cuales la longitud del campo de datos sea 0).

El campo **Checksum** es opcional y no es preciso utilizarlo; un valor 0 en este campo significa que no ha sido calculado. Los diseñadores eligieron hacer este campo opcional para permitir a las implementaciones operar con una sobrecarga computacional mínima

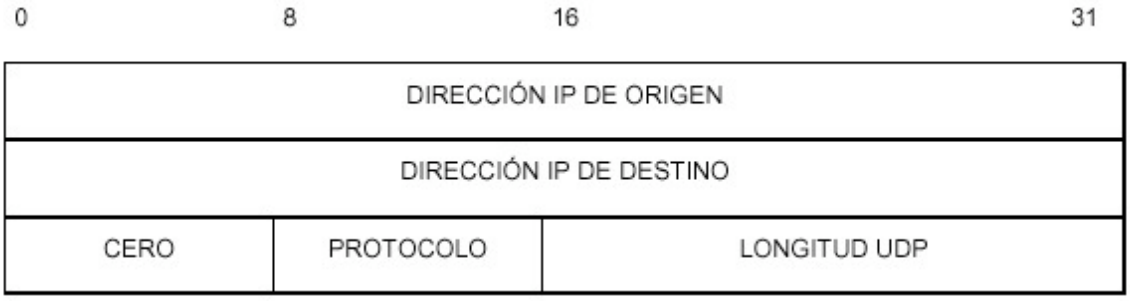
cuando se utiliza UDP a través de una red de área local de alta fiabilidad. Hay que tener en cuenta, sin embargo, que IP no calcula el checksum de la parte de datos del datagrama IP y que por lo tanto, el checksum UDP es la única forma de garantizar que los datos hayan llegado intactos y por lo tanto resulta recomendable su uso.

El cálculo del checksum UDP es similar al que se efectúa en IP: divide los datos en campos de 16 bytes y calcula el complemento a uno de su suma en complemento a uno. Dado que la aritmética de complemento a uno tiene dos representaciones para el 0: todos los bits puestos a 0 o todos los bits puestos a 1, se ha empleado esta segunda representación para el resultado cero al calcular el checksum y la representación con todos los bits a 0 cuando el checksum no se ha calculado.

## PSEUDO-CABECERA UDP

El checksum UDP abarca más información de la contenida en el datagrama UDP. Para calcular el Checksum, UDP prepara una pseudo-cabecera que añade al datagrama antes de efectuar el cálculo, además, rellena el campo de datos con un octeto de ceros cuando es necesario para que la longitud del mensaje sea múltiplo de 16 bits. Esta pseudo-cabecera y el carácter de relleno no se transmiten junto con el datagrama UDP, y están incluidos dentro de su longitud. Para calcular un Checksum, UPD almacena en primer lugar ceros en el campo Checksum, calcula la suma en complemento a uno de 16 bits del datagrama completo, incluyendo la pseudo-cabecera, la cabecera UDP y el campo de datos de usuario.

El objetivo de utilizar esta pseudo-cabecera es verificar que el datagrama UDP ha llegado a su destino correcto. La clave para entender la utilización de la pseudo-cabecera estriba en darse cuenta que el destino correcto consiste en un sistema específico y un puerto de protocolo específico dentro de dicho sistema. La cabecera UDP especifica únicamente el número de puerto de protocolo, por lo tanto, para verificar que el destino es correcto, UDP calcula en el sistema emisor un Checksum que compruebe la dirección de destino IP además del datagrama UDP. En el destinatario, UDP verifica el Checksum utilizando la dirección de destino IP obtenida de la cabecera del datagrama IP que contenía el datagrama UDP. Si el Checksum es correcto, el datagrama habrá alcanzado el sistema de destino deseado así como el puerto de protocolo correcto dentro dicho sistema.




La pseudo-cabecera utilizada en el cálculo del Checksum UDP consta de 12 octetos con la estructura indicada en la figura. Los campos de la pseudo-cabecera, denominados **Dirección IP de origen** y **Dirección IP de destin**o, contienen respectivamente las direcciones IP de origen y destino que deben usarse cuando se envíe el mensaje UDP. El campo **Protocolo** contiene el código del protocolo UDP contenido en la cabecera del

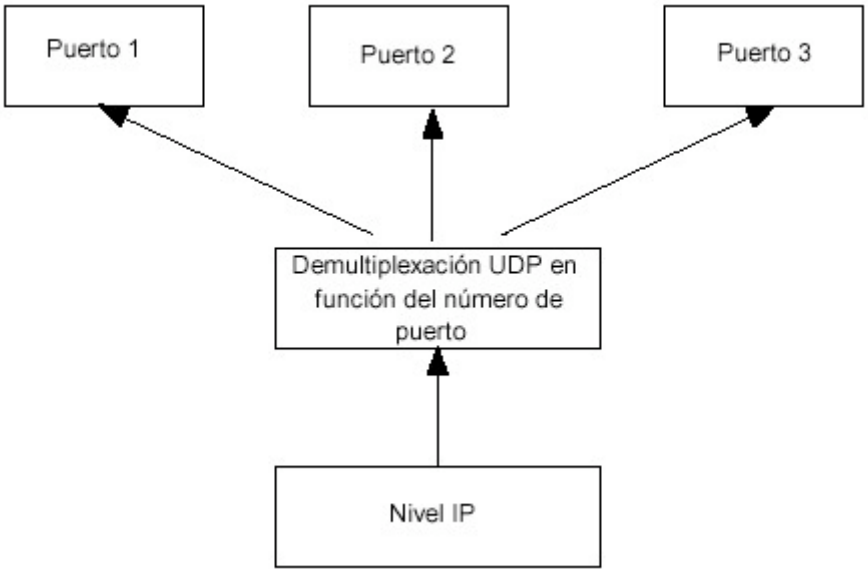
datagrama IP (17 para UDP), y el campo **Longitud de UDP** contiene la longitud del datagrama UDP ( sin incluir la pseudo-cabecera ). Para comprobar el Checksum, el receptor debe extraer estos campos a partir de la cabecera IP, ensamblarlos en el formato de pseudo-cabecera y recalcular el Checksum.

## MULTIPLEXACIÓN DEMULTIPLEXACIÓN Y PUERTOS

En una arquitectura de niveles, como la de TCP/IP, un protocolo debe multiplexar o demultiplexar entre varias entidades del siguiente nivel. UDP acepta datagramas procedentes de múltiples programas de aplicación y los pasa a IP para su transmisión, igualmente, acepta datagramas que le entrega el protocolo IP para entregarlos a su vez al programa de aplicación correspondiente.

Conceptualmente, toda la multiplexación y demultiplexación entre UDP y los programas de aplicación se produce mediante el mecanismo de puerto. En la práctica, cada programa de aplicación debe negociar con el sistema operativo para conseguir un puerto de protocolo y un número de puerto asociado al mismo antes de poder enviar un datagrama UDP. Una vez que el puerto le ha sido asignado cualquier datagrama que el programa de aplicación envía a través de dicho puerto contendrá en el campo **Puerto de Origen** dicho número de puerto asignado.

En cuanto a los datagramas recibidos, UDP aceptará datagramas procedentes de IP y los demultiplexará basándose en el puerto de destino UDP, tal y como muestra la figura.




Como ya hemos dicho, el modo más fácil de pensar en los puertos UDP es en la forma de una cola.

En la mayoría de las implementaciones, cuando un programa de aplicación negocia con el sistema operativo la utilización de un determinado número de puerto, el sistema operativo crea una cola interna para almacenar los mensajes que llegan; generalmente, la aplicación puede especificar o cambiar el tamaño de la cola. Cuando UDP recibe un datagrama, comprueba que el número de puerto de destino se ajusta a uno de los puertos que están actualmente en uso. Si no es así, envía un mensaje de error ICMP de **puerto**

**no alcanzable** y descarta el datagrama. Si se encuentra el número de puerto de destino, UDP encola el datagrama en dicho puerto donde el programa de aplicación puede acceder a él. Desde luego, se produce un error si el puerto está lleno, y UDP descartará igualmente el datagrama.

## NÚMEROS DE PUERTO UDP RESERVADOS Y DISPONIBLES

¿ Cómo debe asignar su número de puerto de protocolo? El problema es importante porque los sistemas necesitan estar de acuerdo en los números de puerto a utilizar antes de que puedan interactuar. Por ejemplo, cuando un sistema desea obtener un fichero contenido otro, necesita saber qué puerto utiliza el programa de transferencia de ficheros del segundo sistema. Existen dos posibilidades en la problemática de asignación de puertos.

La primera es utilizar una autoridad central que asigne los números de puerto, publicando una lista de todas las asignaciones, así todos los programas serán desarrollados de acuerdo a esta lista. Este enfoque se denomina generalmente **asignación universal** y la asignación de puertos especificada por la autoridad es conocida como **asignación de puertos "bien conocidos**".

El segundo enfoque en la asignación de puertos utiliza una **asignación dinámic**a. En la asignación dinámica, los números de puerto no son conocidos de forma global; en su lugar, siempre que un programa necesita un número de puerto el sistema operativo le asigna uno. Para que un sistema conozca la asignación de puertos en otro será necesario que pregunte a aquél acerca de su asignación de puertos.

Los diseñadores de TCP/IP adoptaron un enfoque híbrido en el cual se asignaron unos pocos números de puerto a priori, dejando la mayor parte de ellos para ser asignados a programas de aplicación locales. Los números de puerto asignados comienzan por los valores más bajos y se extienden de forma creciente, dejando los valores más altos libres para una asignación dinámica. En la tabla adjunta se indican algunos de los números de puerto asignados en la actualidad, la segunda columna contiene las palabras claves utilizadas en Internet y la tercer las palabras claves utilizadas en la mayor parte de los sistemas UNIX.

| | **CLAVE** | **CLAVEUNIX** | **DESCRIPCION** |
|---|---|---|---|
| 0 | - | - | RESERVADO |
| 7 | ECHO | echo | ECHO |
| 9 | DISCARD | discard | Discard |
| 11 | USERS | systat | Active Users |
| 13 | DAYTIME | daytime | Daytime |
| 15 | - | nestat | Who is up or NEASTAT |
| 17 | QUOTE | qotd | Quote of the day |
| 19 | CHARGEN | chargen | Character Generator |
| 37 | TIME | time | Time |
| 42 | NAMESERVER | name | Host Name Server |
| 43 | NICNAME | whois | Who Is |
| 53 | DOMAIN | nameserver | Domain Name Server |
| 67 | BOOTPC | bootps | Bootstrap Protocol Server |

| | | | |
|---|---|---|---|
| 68 | BOOTPC | bootpc | Bootstrap Protocol Client |
| 69 | TFTP | tftp | Trivial File Transfer |
| 111 | SUNRPC | sunrpc | Sun Microsystems RPC |
| 123 | NTP | ntp | Network Time Protocol |
| 161 | - | snmp | SMNP net monitor |
| 162 | - | snmp-trap | SNMP traps |
| 512 | - | biff | UNIX comsat |
| 513 | - | who | UNIX rwho daemon |
| 514 | - | syslog | system log |
| 525 | - | timed | Time daemon |

## TAMAÑO MÁXIMO DE LOS DATAGRAMAS UDP

Teóricamente, el tamaño máximo de un datagrama IP es de 65535 bytes, impuesto por el campo longitud total de 16 bits de la cabecera IP. Con un cabecera IP de 20 bytes y una cabecera UDP de 8 bytes, esto deja un máximo de 65507 bytes para el campo de datos de usuario de un datagrama UDP. La mayoría de las implementaciones, sin embargo, sólo permiten cantidades menores de este valor.

Hay dos límites que podemos encontrar. En primer lugar, el programa de aplicación puede estar limitado por su Interfaz de programación. Los API de los Sockets proporcionan una función a la que puede invocar la aplicación para fijar el tamaño del buffer de emisión y recepción. Para un Socket UDP, este tamaño está relacionado directamente con el tamaño máximo de los datagramas UDP que la aplicación puede leer o escribir. La mayoría de los sistemas en la actualidad proporcionan un valor por defecto para el tamaño máximo para los datagramas UDP que pueden ser leídos o escritos de 8192 bytes ( este valor está relacionado con el hecho de que NFS lea y escriba cantidades de datos igual a 8192 bytes).
La siguiente limitación procede de la propia implementación de TCP/IP, puede haber implementaciones que limiten el tamaño del datagrama IP a menos de 65535 bytes. Todo sistema que opere con IP debe ser capaz de recibir al menos datagramas IP de 576 bytes. Es por ello que muchas aplicaciones UDP están diseñadas para restringir los datos de aplicación a 512 bytes o menos, para permanecer por debajo del límite anterior. Este límite de 512 bytes puede encontrarse en protocolos tales como RIP, DNS, TFTP, BOOTP y SNMP.