2 El protocolo TCP

2.1 INTRODUCCIÓN

En primer lugar analizaremos las características fundamentales del protocolo de transporte TCP basándonos tanto en la propia definición del protocolo establecida en el RFC 793 como en posteriores ampliaciones consistentes en la introducción de diversos mecanismos para mejorar su rendimiento [Jac88, RFC1323, RFC2581].

TCP es un protocolo de transporte orientado a conexión enormemente extendido en Internet. Las aplicaciones de red más populares (*ftp*, *telnet*, acceso *Web*…) lo utilizan en sus comunicaciones.

La función principal del nivel de transporte dentro de la arquitectura de protocolos TCP/IP es la de permitir la comunicación extremo a extremo entre dos aplicaciones de forma económica y fiable. La unidad básica de transferencia se denomina segmento, de tamaño máximo el denominado MSS (*Maximum Segment Size*) expresado en octetos, que como veremos más adelante se negociará por los extremos de la comunicación en el establecimiento de la misma.

Existe otro protocolo de transporte en la arquitectura TCP/IP muy diferente, UDP (*User Datagram Protocol*). Éste es mucho más sencillo que TCP. Se limita a enviar paquetes de datos, denominados *datagramas,* de un terminal a otro sin garantizar que éstos sean recibidos correctamente. Si la aplicación requiere fiabilidad en la comunicación, deberá ser ella misma la que se la proporcione o bien se tendrá que recurrir al TCP.

2.2 CARACTERÍSTICAS GENERALES DE TCP

TCP es un protocolo que proporciona un servicio de transporte de datos que ofrece al nivel superior:

- Fiabilidad
- Control de Flujo
- Orientación a conexión
- Multiplexación
- Orientación a flujo de octetos
- Transferencia con almacenamiento

Aunque en la definición de TCP no aparece ningún mecanismo específico para el ***control de la congestión***, varios son los algoritmos desarrollados posteriormente con este objetivo, y los cuales también detallaremos en este capítulo.

A continuación analizaremos cada una de estas características.

- **Transmisión fiable**

TCP está diseñado para recuperarse ante situaciones de corrupción, pérdida, duplicación o desorden de datos que puedan generarse durante el proceso de comunicación. Para conseguirlo, utiliza reconocimientos positivos y retransmisiones. Cada octeto de datos transmitido tiene asignado un número de secuencia. El número de secuencia del primer octeto de datos en un segmento se almacena en la cabecera del mismo y recibe el nombre de Número de Secuencia del Segmento. Los segmentos también contienen un número de reconocimiento que identifica el número de secuencia del siguiente octeto que se espera recibir.

Cuando TCP transmite un segmento con datos, coloca una copia en la cola de retransmisión e inicializa un temporizador. Al recibir el reconocimiento (ACK) para él, TCP lo borra de la cola. Si no se llega a recibir el ACK antes de que el temporizador expire, el segmento es retransmitido.

En recepción, los números de secuencia son utilizados para ordenar correctamente los segmentos (en caso de que alguno llegue fuera de orden) y para eliminar los duplicados. La corrupción de segmentos a nivel de transporte se detecta a través del *Checksum* incluido en cada uno de ellos y el cual es verificado en recepción. Todo segmento erróneo es descartado inmediatamente y no da lugar a reconocimiento alguno.

TCP utiliza un esquema de **reconocimientos acumulativos**, es decir, el receptor informa con el número de reconocimiento de hasta qué octeto del flujo de datos enviados ha recibido correctamente. Este sistema presenta varias ventajas:

- Por un lado los reconocimientos son fáciles de generar y no resultan ambiguos.
- Por otro, la pérdida de reconocimientos no fuerza, necesariamente, la retransmisión de segmentos.

Esta característica de TCP permitirá mejorar su comportamiento en **enlaces asimétricos**, tal y como se verá en el capítulo correspondiente.

Sin embargo, el carácter acumulativo de los reconocimientos hace que el emisor no reciba información de todas las transmisiones correctas sino únicamente del último octeto de flujo continuo recibido sin errores.

El tratamiento de la temporización y la retransmisión es fundamental en TCP**, tanto en entornos como el móvil en los que las pérdidas son frecuentes, como en redes fijas en las cuales la congestión puede dar lugar a pérdidas de paquetes o retardos importantes.**

Para llevar a cabo este tratamiento se han establecido las siguientes estrategias:

a) Algoritmo de retransmisión adaptativo [Ste94, Jac88]

El ajuste del temporizador de retransmisión, RTO (*Retransmission Time Out*), es especialmente crítico en TCP, al actuar tanto en redes locales, en enlaces punto a punto o en una red tan cambiante como Internet. Debe asegurarse un mecanismo que funcione correctamente en entornos tan diferentes como en los que opera TCP. RTO debe ser suficientemente pequeño como para responder rápidamente a las pérdidas, pero no tanto como para forzar la retransmisión de datos que han sufrido un pico de retardo en la red sin haber llegado a perderse, como sería el caso de congestión.

Para adaptarse a los retardos variables característicos de un entorno como Internet, TCP usa un algoritmo de retransmisión adaptativo que monitoriza el retardo en cada conexión y ajusta el valor de RTO de acuerdo con ese valor. La especificación del protocolo sugiere tomar muestras del tiempo de ida y vuelta, RTT (*Round Trip Time*), calculado como la diferencia de tiempo entre la emisión de un segmento y la recepción de su reconocimiento. Con esta información, TCP puede ajustar dinámicamente una variable que identifique el tiempo medio de ida y vuelta, de la siguiente forma:

$$RTT = \alpha * RTT_{anterior} + (1-\alpha) * RTT_{nuevo}$$

Finalmente el tiempo de retransmisión, se ajusta a:

$$RTO = \beta * RTT$$

Los valores recomendados de α y β son 0.9 y 2 respectivamente, para el caso de redes fijas. Dado que estos valores han sido hallados experimentalmente, no se asegura el buen funcionamiento del algoritmo en circunstancias muy particulares.

Esta estrategia, no obstante, no se adapta a fluctuaciones importantes en RTT provocando retransmisiones innecesarias. Se hace necesario introducir una estimación, también, de la varianza del tiempo de ida y vuelta, por lo tanto:

$$RTO = a + 4 * d$$

a: RTT medio

d: estimador de la desviación media de RTT

Cada vez que se obtiene una nueva muestra del tiempo de ida y vuelta, m, los estimadores se actualizan así:

$$Err = m - a$$

$$a = a + g * Err$$

$$d = d + h * (|Err| - d)$$

Los valores recomendados para g y h son 0.125 y 0.25 respectivamente, también obtenidos experimentalmente.

b) Algoritmo de *Karn [KaP87]*

Cuando se recibe un reconocimiento de un segmento que ha sido objeto de retransmisión, es imposible determinar si éste corresponde al primer segmento transmitido o a su posterior retransmisión. El algoritmo de *Karn* evita esta ambigüedad.

Este algoritmo establece que el cálculo del RTT estimado para determinar RTO debe hacerse ignorando las muestras correspondientes a segmentos retransmitidos.

c) Marca Temporal o *Timestamp*

Existe una solución alternativa para la medida exacta de RTT que hace desaparecer la ambigüedad, de forma que hace innecesaria la aplicación del algoritmo de *Karn*. Esta solución se basa en aprovechar el ancho de banda para mandar la información de tiempo en cada segmento. De esta forma puede realizarse una medida por cada segmento

enviado independientemente de si se trata de un segmento retransmitido o no, obteniendo así más medidas. Esta solución, si **bien es adecuada para redes de alta velocidad** en las que el ancho de banda no es un recurso escaso, **no lo es para transmisiones sobre el canal móvil** en las que el ancho de banda es un recurso que debe gestionarse y utilizarse de forma eficiente.

d) *Backoff* exponencial

Cuando se producen retransmisiones, según el RFC 793 debe aplicarse el RTO que exista en ese momento. No obstante, en caso de congestión esta política es contraproducente, ya que la propia congestión produce incrementos en el RTT. Consecuentemente se producirán retransmisiones innecesarias, ya que debido a la ambigüedad en la medida de RTT, las medidas de paquetes retransmitidos no se tienen en cuenta, y por lo tanto el valor de RTO no se actualizará. Para evitar esta situación, la mayoría de implementaciones incorporan el algoritmo de *backoff* exponencial.

Se establece un mecanismo de *backoff* exponencial para espaciar las retransmisiones consecutivas de un mismo paquete, dando así tiempo a la red para que resuelva la congestión. La estrategia de *backoff* exponencial utiliza, inicialmente, el tiempo de retransmisión calculado a partir de las fórmulas vistas anteriormente. Sin embargo, si el temporizador expira y se produce una retransmisión, TCP incrementa el tiempo de retransmisión. Si las retransmisiones persisten, TCP incrementa sucesivamente el RTO (para evitar un aumento excesivo de este parámetro, la mayoría de implementaciones de TCP limitan su valor a un máximo que supera el mayor retardo posible en Internet.). Se utiliza el *backoff* hasta que llega confirmación de un paquete que no ha sido retransmitido

Aunque existen implementaciones que utilizan técnicas diferentes para establecer el *backoff*, la mayoría utilizan un factor multiplicativo:

$$RTO_{nuevo} = ß\ RTO_{anterior} \qquad \text{con} \qquad ß = 2 \text{ (normalmente)}$$

Ya que las pérdidas de paquetes son interpretadas por TCP como síntoma claro de congestión en la red, esta estrategia **ayuda a frenar la inyección de datos**, en estos casos, mejorando así la situación en la red. Ahora bien, en **entornos móviles**, en el que los errores no tienen su origen en la congestión del enlace sino, fundamentalmente, en el propio carácter errático del canal de comunicación, tanto éste como el resto de mecanismos de control de la congestión que incorpora TCP resultan muy poco idóneos ya que **contribuyen a retrasar la detección y recuperación** del protocolo ante los errores.

En particular, este algoritmo puede provocar una pérdida considerable de eficiencia del protocolo en casos de ráfagas de errores o de desconexiones temporales. En estos casos es muy probable que tras repetidas retransmisiones el valor de RTO llegue a su valor máximo ($RTO_{máx}$), de forma que una vez se restablezca la comunicación, no sea tras $RTO_{máx}$ que se

retransmita el primer segmento afectado por la desconexión. Esto produce retardos muy elevados, ya que este valor máximo **está adecuado a los casos de congestión**.

Existen también implementaciones de TCP en las que la retransmisión se realiza de forma selectiva [FaF96, RFC2018]. De esta forma se solucionan los problemas que comporta la confirmación positiva, evitando retransmisiones innecesarias, ya que en este caso el emisor conoce con exactitud los segmentos recibidos en el extremo receptor. La retransmisión selectiva implica complejidad en el protocolo y solamente será justificable en aquellos casos en los que haya múltiples pérdidas por ventana de transmisión y las ganancias de un método respecto al otro sean considerables.

- **Control de Flujo**

TCP es un protocolo de ventana deslizante. Este mecanismo surge como mejora del usado por protocolos con reconocimientos positivos de tipo *Stop&Wait*. Estos protocolos obligan al emisor a retrasar la emisión de cada nuevo paquete hasta que se recibe el ACK del anterior, desaprovechando así, la posible capacidad de comunicación bidireccional de la red. Los protocolos de ventana deslizante aprovechan mejor el ancho de banda al permitir transmitir un número determinado de paquetes antes de que llegue el ACK correspondiente. La ventana se coloca sobre la secuencia de octetos que configuran el flujo de datos proveniente de la aplicación e indica qué paquetes pueden ser transmitidos.

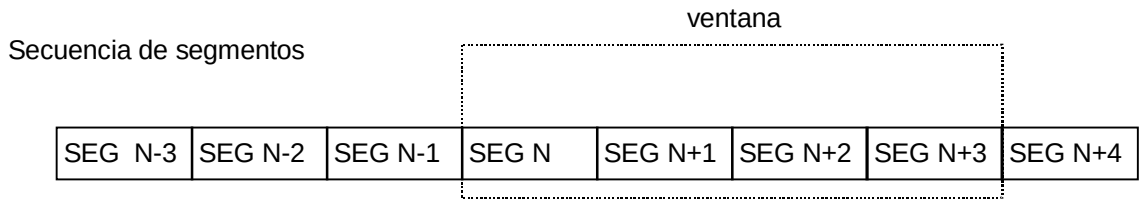


Figura 2.1 Protocolo de ventana deslizante

El número de paquetes no reconocidos es como máximo, en cada momento, igual al tamaño de la ventana. Si el protocolo está bien “sintonizado”, mantendrá el enlace completamente saturado.

El valor óptimo para optimizar el ancho de banda disponible (BW) es:

$$\text{ventana} = RTT * BW$$

TCP utiliza un mecanismo de ventana deslizante especial que le sirve tanto para conseguir una transmisión eficiente (el emisor puede enviar múltiples segmentos sin esperar su reconocimiento) como para proporcionar control de flujo permitiendo al receptor restringir el número de octetos que puede recibir en cada momento. El receptor envía, junto con cada reconocimiento, el tamaño de ventana que puede aceptar en ese momento, es decir, el rango de números de secuencia aceptables a partir del último segmento recibido correctamente. Este tamaño puede variar a lo

largo de la conexión (incluso llegando a valer 0). El emisor aplica continuamente la ventana recibida para determinar qué paquetes puede enviar.

- **Multiplexación**

Permite que varios procesos de una misma máquina utilicen simultáneamente el servicio que ofrece TCP. Éstos se diferencian dentro de la misma máquina por el valor del puerto asignado.

El protocolo proporciona una dirección o puerto a cada aplicación que lo usa. El conjunto formado por un número de puerto (que identifica una aplicación en una máquina) y una dirección IP (que identifica una máquina) recibe el nombre de *socket*.

La asignación de puertos a procesos es manejada de forma independiente por cada máquina. No obstante, es común asignar números de puerto universalmente conocidos a algunos servidores de aplicaciones estándar sobre TCP. Algunos de ellos son [RFC1700]:

Puerto	**Nombre**	**Descripción**
7	*echo*	*echo protocol*
9	*discard*	*discard protocol*
11	*systat*	*active users protocol*
13	*daytime*	*daytime protocol*
17	*qotd*	*quote of the day protocol*
19	*chargen*	*character generator protocol*
20	*ftp-data*	*datos de file transfer protocol*
21	*ftp*	*file transfer protocol*
23	*telnet*	*conexión de terminal remoto*
37	*time*	*network time protocol*
69	*tftp*	*trivial file transfer protocol*
161	*snmp*	*simple network management protocol*

- **Comunicación orientada a conexión**

Los mecanismos que utiliza TCP para proporcionar fiabilidad, control de flujo y control de congestión requieren que el protocolo inicialice y mantenga cierta información sobre el estado del flujo de datos. La combinación de toda esta información recibe el nombre de **conexión**. Cada conexión está unívocamente identificada por un par de *sockets* que identifica a los dos extremos de la comunicación.

Cuando dos procesos quieren comunicarse, sus TCP deben establecer, primero, una conexión (es decir, inicializar la información de estado en cada extremo). Cuando la comunicación se ha completado, la conexión se cierra liberando los recursos para otros usos posteriores.

TCP **inicia** la conexión mediante un intercambio de mensajes de sincronismo (SYN) a tres bandas (*three way handshake)*. Primero el extremo que quiere iniciar la conexión envía un mensaje SYN. En caso de que el extremo remoto acepte la conexión, manda a su vez un mensaje SYN que además confirma al mensaje anterior. Finalmente, el que ha iniciado la conexión manda a su vez un mensaje de confirmación (ACK). De esta forma los dos extremos

de la conexión se aseguran que ésta puede establecerse, es decir, que ambos extremos están de acuerdo.

Además, este intercambio de mensajes a tres bandas permite a ambos extremos acordar un número de secuencia inicial. Éstos se seleccionan de forma aleatoria, y se usarán para identificar los octetos en el flujo de datos que se envía. Se trata del **sincronismo de la conexión**.

Una vez iniciada la conexión, empieza el período de intercambio de datos hasta que el programa de aplicación informa a TCP que ya no tiene más datos para enviar. Para cerrar la conexión en cada uno de los sentidos se termina de transmitir los datos restantes y se envía un segmento de finalización (FIN). El otro extremo reconoce el segmento FIN y notifica a la aplicación de que ya no existen más datos disponibles. Aunque se haya cerrado la conexión en uno de los sentidos, ésta todavía puede permanecer activa en el otro.

La Figura 2.2 muestra estos intercambios de mensajes. En ella, además, se identifican los diferentes estados en los que se encuentra el TCP. En el Anexo A se muestra el Diagrama de la Máquina de estados de TCP.

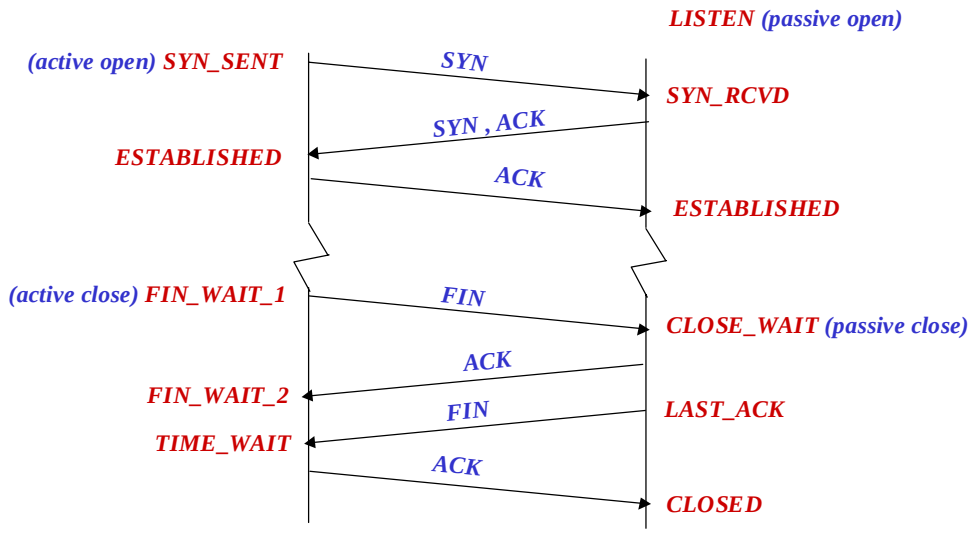


Figura 2.2 Intercambio de mensajes para el inicio y el cierre de la conexión

- **Orientación a flujo de octetos**

El volumen de información transferido entre dos aplicaciones que utilizan TCP como protocolo de transporte consiste en un flujo de octetos sin ningún tipo de marcas insertadas por TCP para indicar formato o estructura alguna.

Si, en un extremo de la conexión, la aplicación lleva a cabo una escritura de 10 octetos seguida de otra de 20 octetos y otra de 30 octetos, la aplicación del otro extremo no tiene forma alguna de determinar el tamaño concreto de cada una de ellas ya que se limita a recibir un flujo de 60 octetos sin marca o indicación alguna (es decir, quizá lo reciba a través de 3 lecturas de 20 octetos o 6

lecturas de 10 octetos, etc.). Un extremo inyecta un flujo de octetos en el nivel de transporte y el otro recibe exactamente la misma secuencia de octetos.

- **Transferencia con almacenamiento**

Los programas del nivel de aplicación envían un flujo de datos a través del circuito virtual establecido entre los dos extremos, entregando continuamente octetos de información al *software* del protocolo. Al transferir los datos, cada aplicación utiliza fragmentos del tamaño que considera adecuado. TCP puede almacenar el número apropiado de octetos que permita, posteriormente, generar el datagrama de tamaño adecuado para ser transmitido por la red.

Esto significa que, aunque la aplicación genere bloques de datos de tamaño muy reducido, TCP puede unirlos y permitir una transmisión más eficiente. De igual manera, si la aplicación genera bloques de datos muy grandes, el protocolo puede decidir dividirlos.

2.3 CONTROL DE LA CONGESTIÓN

Cuando se definió e implementó TCP, las redes existentes presentaban como problema principal una baja fiabilidad, es decir, la presencia de errores era la característica limitante del comportamiento eficiente de la red. Las situaciones de congestión, causa principal del deterioro del comportamiento de las redes actuales, no fueron tenidas en cuenta y, por ello, no se especificó mecanismo alguno para su control. Con el tiempo, sin embargo, se han ido desarrollando una serie de algoritmos con ese propósito [Jac88, RFC2581]:

a) Inicio lento *o Slow Start*

Esta estrategia se basa en el siguiente principio de equilibrio en la conexión: "*la tasa a la cual deberían inyectarse nuevos paquetes en la red es la tasa a la que llegan nuevos reconocimientos*". Esta afirmación indica que TCP es un protocolo autosincronizado ya que utiliza los reconocimientos como marcas para inyectar nuevos paquetes en la red. Cuando no hay segmentos en tránsito, como en el inicio de una conexión o tras una expiración del temporizador de retransmisión, no existen ACK que permitan activar tal comportamiento; para conseguir que los paquetes fluyan son necesarios ACK que lo permitan y para tener ACK es necesario un flujo de paquetes.

El mecanismo de Inicio Lento permite incrementar gradualmente la cantidad de datos en tránsito. Es el impulso inicial necesario para conseguir llevar la conexión al estado de equilibrio.

Este mecanismo utiliza una nueva ventana llamada ventana de congestión (*cwnd*) de manera que en cada momento, el emisor puede enviar el mínimo número de segmentos de entre la cantidad permitida por la ventana de control de flujo y la permitida por la ventana de congestión. Esto significa que el algoritmo de Inicio Lento dejará de tener efecto una vez que *cwnd* alcance el

tamaño de la ventana del receptor (siempre y cuando no se haya producido antes una pérdida de paquetes que inicie de nuevo el algoritmo).

El algoritmo opera de la siguiente forma:

- Al iniciarse una nueva conexión o al reiniciarse debido a una pérdida, inicializa *cwnd* a un segmento:

  cwnd=1

- Cada vez que se recibe un ACK incrementa en una unidad *cwnd:*

  cwnd++

Si, por ejemplo, el receptor envía un reconocimiento por cada segmento recibido, el emisor enviará 1 segmento durante el primer RTT, 2 segmentos durante el segundo, 4 durante el tercero y así sucesivamente. Esto provoca un **incremento exponencial** de la ventana, ya que con cada reconocimiento recibido el emisor puede enviar dos paquetes: uno debido al propio ACK (la ventana se desliza un segmento hacia la derecha) y otro por la apertura de la ventana exigida por el mecanismo de *Inicio Lento*.

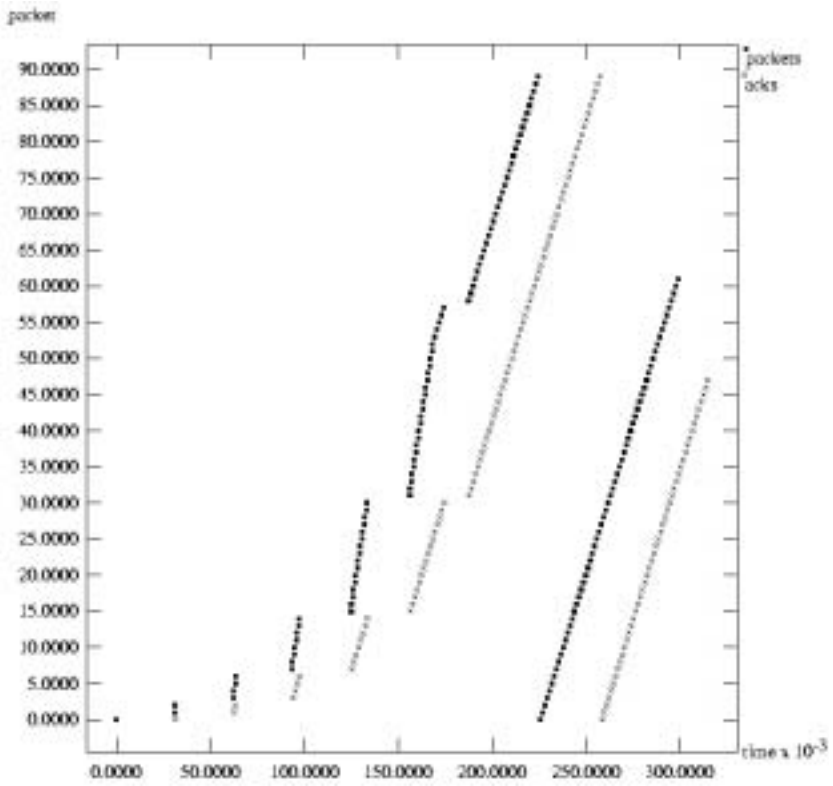


Figura 2.3 Flujo de paquetes y reconocimientos

Respecto a este inicio lento hay propuestas [RFC2414, RFC2415] para solucionar los inconvenientes que este mecanismo tiene en el inicio de la comunicación, permitiendo una

apertura más rápida de la ventana de congestión. Las técnicas propuestas tienen importancia en transferencias cortas en las que la fase de inicio es relevante en la comunicación.

La Figura 2.3 muestra la evolución del número de secuencia (*packets*) y de reconocimiento (*acks*) módulo 90 (expresado en segmentos), de una transferencia *ftp* sobre un enlace de 10 Mbps, un retardo de propagación de 15 milisegundos, y un tamaño de paquete de 1500 octetos. En estas condiciones el tamaño de ventana óptimo para la transmisión es de 27 segmentos. Puede observarse que al inicio de la trasferencia, el crecimiento de la ventana es exponencial (correspondiente a la fase de inicio lento). Una vez se ha alcanzado el tamaño máximo de ventana, se mantendrá su valor máximo hasta que se produzca una retransmisión.

b) Algoritmo de Prevención de la Congestión ó *Congestion Avoidance*

Este algoritmo persigue adaptar la transferencia a la situación de la red en cada momento, reaccionando ante posibles estados de congestión.

Puesto que TCP asume que las pérdidas de paquetes son debidas a congestión, TCP interpreta una expiración del temporizador de retransmisión (*timeout*) como un síntoma claro de congestión. Cuando esto ocurre, se lleva a cabo un estrechamiento exponencial de la ventana que consiste en reducir su tamaño a la mitad. Por contra, cuando la red no pierde paquetes, este algoritmo propone un ensanchamiento lineal de la ventana que permita la adaptación progresiva a la nueva situación descongestionada aprovechando al máximo el ancho de banda disponible.

El algoritmo opera de la siguiente manera:

- Cuando ocurre un *timeout*, asigna a *cwnd* la mitad del tamaño de la ventana actual:

  $$cwnd = W/2$$

  Este es el **estrechamiento exponencial** ya comentado.

- Cada vez que llega un reconocimiento de nuevos datos, incrementa *cwnd* en 1/*cwnd* (es decir, se incrementa 1 segmento por ventana completa transmitida):

  $$cwnd = cwnd + 1/cwnd$$

  Este es el **ensanchamiento lineal** ya comentado. Teniendo en cuenta que el número de ACK que puede recibir el receptor durante el tiempo de ida y vuelta es, como máximo, *cwnd* (donde *cwnd* es el tamaño en segmentos de la ventana de congestión), el incremento máximo en el tamaño de la ventana durante un RTT será de un segmento.

c) Combinación de *Slow Start* y *Congestion Avoidance*

Inicio Lento y Prevención de la Congestión son algoritmos totalmente independientes, no obstante, en la práctica, se implementan de manera conjunta.

El algoritmo combinado mantiene dos variables para el control de la congestión: una ventana de congestión (*cwnd*) y un valor umbral (*ssthresh*) que permite conmutar entre los dos algoritmos. Su funcionamiento es el siguiente:

- Inicialmente:

  cwnd = 1

  ssthres = W_{max} (tamaño máximo de la ventana de transmisión)

- Cuando ocurre un *timeout*, la mitad del tamaño actual de la ventana (y como mínimo dos segmentos) se guarda en la variable *ssthresh*:

  ssthresh=W/2

  Este es el estrechamiento exponencial típico de Prevención de la Congestión.

- Asigna a *cwnd* un único segmento:

  cwnd=1

  Esto indica el comienzo de Inicio Lento.

- La llegada de un ACK que reconoce datos nuevos, incrementa el valor de *cwnd* según el siguiente procedimiento:

  - Si ***cwnd*** $\leq$ ***ssthresh*** significa que estamos en Inicio Lento por lo que incrementamos *cwnd* en una unidad.

  - Si ***cwnd*** $>$ ***ssthresh*** significa que estamos en Prevención de la Congestión por lo que incrementamos *cwnd* en 1/*cwnd*. El emisor siempre envía el mínimo entre *cwnd* y la ventana de control de flujo o transmisión.

La Figura 2.4 muestra la evolución de la ventana de congestión para los algoritmos de Inicio Lento y Prevención de la Congestión. En este ejemplo, el umbral de congestión se encuentra en 32 segmentos (línea discontinua), a partir del cual se pasa de la fase de Inicio Lento (crecimiento exponencial) a la fase de Prevención de la Congestión (crecimiento lineal). También puede apreciarse como tras una retransmisión, el umbral de congestión se reduce a la mitad de la ventana de congestión y se inicia otra vez el proceso.

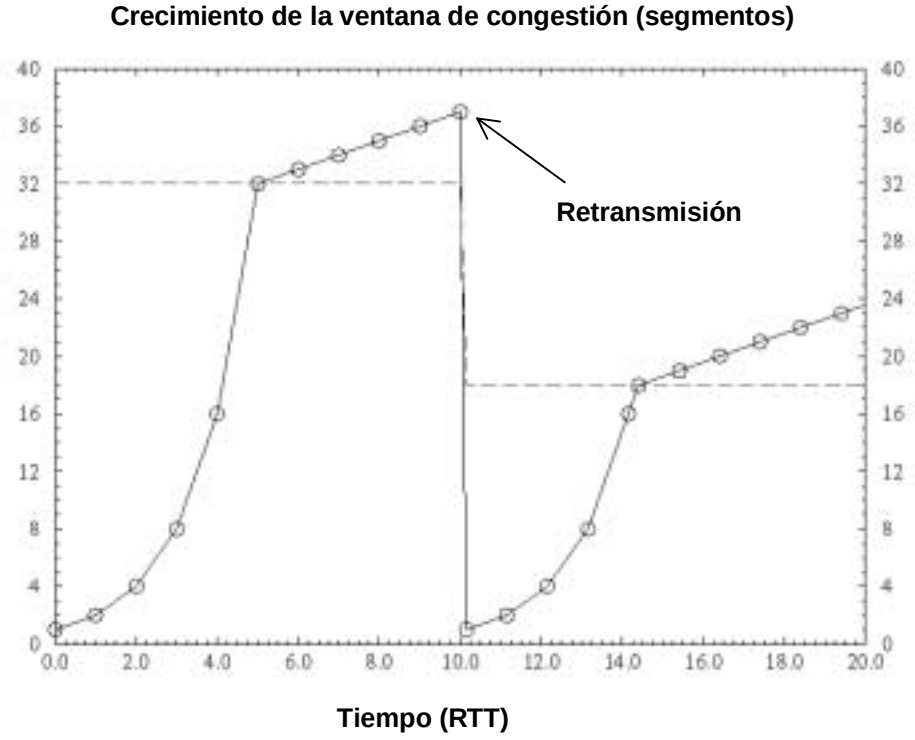


Figura 2.4 Algoritmos de Inicio Lento y Prevención de la Congestión

Los algoritmos de control de la congestión son **adecuados para este tipo de situaciones**, pero **no lo son en el caso de redes móviles** en las que el principal problema es la pérdida de paquetes debido a errores o desconexiones en la comunicación. En las redes móviles interesa que los errores se recuperen de la forma más rápida posible, y sin interactuar con la transmisión, ya que este tipo de errores son ajenos al protocolo. Es decir, el hecho de reducir el flujo de datos que el emisor puede insertar a la red no hace más que retrasar la recuperación de la transmisión y no por ello se reducirá la tasa de pérdidas.

2.4 MEJORAS A LOS MECANISMOS

A medida que las redes y los protocolos van evolucionando, se van incorporado nuevos mecanismos, que tienen el propósito de mejorar el comportamiento del protocolo. En concreto los mecanismos de retransmisión y recuperación rápida detallados a continuación [Ste94, RFC2581], intentan optimizar y ajustar los algoritmos descritos anteriormente para adecuar el protocolo TCP a situaciones de congestión y pérdidas en general, y ayudarlo así a recuperarse más rápidamente de las mismas.

Los mecanismos de congestión actúan a posteriori, es decir, una vez detectada la pérdida o presunta pérdida de un segmento. Esto representa que no se intentará solucionar el problema hasta transcurridos RTO segundos.

La primera de las mejoras propuesta que está presente en la mayoría de implementaciones, aprovecha la recepción de **reconocimientos duplicados** (aquellos que consecutivamente reconocen al mismo segmento, es decir contienen el mismo número de secuencia). De esta forma se activa la retransmisión de un segmento presuntamente perdido antes de que expire su temporizador de retransmisión.

Este algoritmo se conoce como **Retransmisión Rápida** ó ***Fast Retransmit***.

La recepción de un ACK duplicado puede ser debida las siguientes situaciones:

- La red desordena paquetes y, en consecuencia, es posible que el receptor haya enviado un reconocimiento duplicado ante la llegada de un segmento que no sigue la secuencia normal. Hay que tener en cuenta que todas las implementaciones modernas de TCP generan, inmediatamente, un ACK duplicado al recibir un segmento fuera de orden.

- Se ha perdido algún segmento de datos tanto por errores del canal o por problemas de congestión. En este caso el TCP recibe segmentos fuera de orden y en consecuencia genera ACK duplicados.

- Se ha producido un pico de retardo en la red, es decir, el tiempo de ida y vuelta de un paquete se ha incrementado repentinamente provocando la expiración del temporizador de retransmisión del emisor con la consiguiente retransmisión del segmento conflictivo. Puesto que el segmento en cuestión ya había sido recibido correctamente, la recepción de su réplica provoca la generación de un ACK duplicado. Esta situación se producirá básicamente en casos de congestión.

El emisor, sin embargo, no sabe a cuál de estas razones responde la recepción del ACK duplicado. Experimentalmente se ha comprobado que no resulta apropiado precipitar la retransmisión ante la recepción del primer ni el segundo reconocimiento duplicado en redes que desordenan paquetes, ya que puede tratarse de un simple problema de reordenación de rápida solución. Por esta razón, la mayoría de implementaciones de TCP activan el mecanismo de Retransmisión Rápida al recibir el **tercer ACK duplicado**. Cuando éste llega, el emisor no espera a la expiración del temporizador sino que retransmite, inmediatamente, el segmento que demanda el reconocimiento e inicia el algoritmo de Inicio Lento.

Este mecanismo, pensado para situaciones de congestión, es también muy adecuado para las situaciones en las que la pérdida es debida a errores en la comunicación, ya que en el mejor de los casos se retransmitirá el paquete perdido antes que en implementaciones sin este mecanismo**. Es por tanto adecuado tanto para redes fijas como móviles.**

Además de la retransmisión de los datos perdidos, los algoritmos contra la congestión reducen la ventana de transmisión, para dar tiempo a la red que se recupere de ésta.

El algoritmo combinado de Inicio Lento y Prevención de la Congestión da una solución muy drástica a los problemas de congestión, pudiendo degradar de forma innecesaria el comportamiento de TCP frente a estas situaciones. El hecho de cerrar la ventana de congestión a un único segmento después de la pérdida de datos y empezar un inicio lento no parece adecuado ni en los casos de congestión ni en los casos de errores.

Analizando con más detalle el mecanismo propuesto anteriormente, puede ampliarse basándonos en el hecho que un ACK duplicado no sólo indica que ha habido un problema en la red, sino que también confirma que un paquete ha abandonado la red y ha sido recibido correctamente por el TCP en el otro extremo.

Esto es así puesto que el receptor sólo puede generar un ACK como respuesta a la llegada de un paquete. No hay necesidad, por tanto, de poner en marcha Inicio Lento y reducir así drásticamente el número de paquetes inyectados en la red.

Este algoritmo se conoce como Retransmisión Rápida con Recuperación Rápida ó *Fast Retransmit with Fast Recovery*, y opera como sigue:

- Cuando llega el tercer ACK duplicado se retransmite el segmento perdido y:

$$ssthresh = cwnd/2$$

$$cwnd = ssthresh + 3$$

- Con cada nuevo ACK duplicado recibido, se incrementa *cwnd* en una unidad y se transmite un nuevo paquete si lo permite el valor de *cwnd*.

- Cuando se recibe el primer ACK que reconoce nuevos datos, se asigna a *cwnd* el valor de *ssthresh* (es decir la mitad del valor que tenía la ventana de congestión cuando se produjo la congestión).

2.5 FORMATO DEL SEGMENTO TCP

Una vez estudiados los mecanismos que utiliza TCP para la transferencia fiable extremo a extremo, podemos pasar a ver qué campos son necesarios en la cabecera del segmento para poder llevar a cabo todas las funcionalidades.

La unidad de transferencia intercambiada entre los niveles de transporte de los dos extremos de la comunicación, cuando se utiliza TCP, se denomina Segmento.

El intercambio de segmentos permite establecer conexiones, transferir datos, enviar reconocimientos, informar sobre el tamaño de ventana y cerrar conexiones. En TCP únicamente tenemos un tipo de segmento que realizará cada una de las funciones particulares dependiendo del tipo asignado dentro del mismo.

El formato de un segmento TCP se muestra en la Figura 2.5.

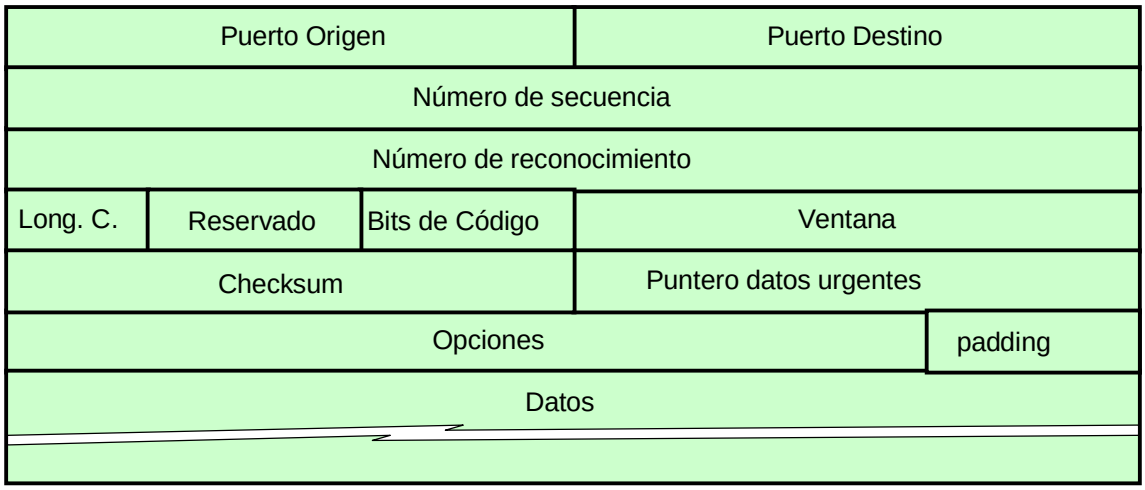


Figura 2.5 Formato del segmento TCP

Como puede verse, cada segmento se divide en una cabecera, que incluye la información de control e identificación, y en un campo de datos.

La cabecera TCP está constituida por los siguientes campos:

- ***Puerto Origen*** (16 bits) : Identifica el puerto emisor.
- ***Puerto Destino*** (16 bits): Identifica el puerto receptor.
- ***Número de Secuencia*** (32 bits): Identifica el octeto, dentro del flujo de datos, al que corresponde el primer octeto del segmento en cuestión.
- ***Número de Reconocimiento*** (32 bits): Indica el número de secuencia que se espera recibir. Su valor es igual al número de secuencia del último octeto recibido más uno. Para que el contenido de este campo sea válido es necesario que la bandera ACK del campo de bits de código, que veremos a continuación, esté activado.
- ***Longitud de Cabecera*** (4 bits): Indica la longitud de la cabecera TCP en palabras de 32 bits. Este campo es necesario ya que el campo Opciones dentro de la cabecera es opcional y de longitud variable. El valor usual es 5, correspondiente a una cabecera de 20 octetos. El valor máximo es 15, es decir, 60 octetos.
- ***Reservado*** (6 bits): Bits de uso reservado.
- ***Bits de código*** (6 bits): Está formado por 6 banderas de 1 bit cada una. Cuando vale 1 representa que la bandera correspondiente está activa. Las diferentes banderas se describen a continuación.

URG ➢ Indica que el campo del puntero urgente es válido. Dado que TCP es un protocolo orientado a flujo, en caso de que se quieran tratar datos urgentes debe especificarse como una excepción. Para este fin esta bandera da la posibilidad de pasar datos urgentes delante de datos antiguos. Un extremo puede marcar los datos como urgentes, TCP debe notificarlo al proceso receptor, y la aplicación debe interpretarlo. Alguna aplicaciones que lo utilizan son *telnet, rlogin y ftp.*

ACK ➢ Indica que el Número de Reconocimiento es válido.

PSH ➢ Indica que el receptor debe pasar los datos del segmento a la aplicación inmediatamente. Es decir, es una notificación del emisor al receptor para que éste pase todos los datos al proceso receptor. Normalmente se usa para comunicaciones interactivas (*telnet, ftp-control*), pero su uso no está restringido a ellas.

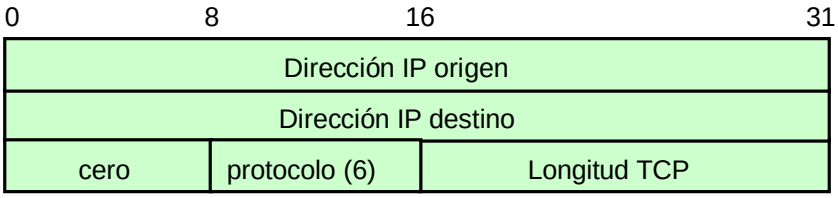
RST ➢ Reinicializa la conexión. Puede ser debido a diferentes problemas como la pérdida de conexión de red. El hecho de reinicializar la conexión permite resincronizar los números de secuencia.

SYN ➢ Permite sincronizar los Números de Secuencia al iniciar una conexión o después de una reinicialización.

FIN ➢ Indica la finalización del envío de datos. Puede estar activo tanto en los segmentos de origen a destino como en los pertenecientes al flujo contrario ya que se trata de una comunicación *full-duplex*.

- ***Ventana*** (16 bits): Tamaño de la ventana de transmisión. Número máximo de octetos que se está dispuesto a aceptar.

- ***Checksum*** (16 bits): Verifica la integridad de los datos y la cabecera. Se calcula usando, además, una pseudo-cabecera que IP pasa a TCP y que incluye la dirección IP origen y destino (esto permite a TCP verificar que el destino es el correcto), el campo de protocolo y el campo de Longitud del segmento.



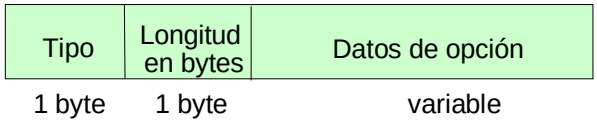
Para el cálculo de éste se utiliza aritmética de 16 bits, es decir, se dividen la pseudo-cabecera y la cabecera en palabras de 16 bits (utilizando relleno cero si es necesario). Se

toma el complemento a uno de la suma y al resultado se le aplica el complemento a 1. En recepción, al calcular el valor de este campo, se considera su valor como cero.

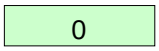
- ***Puntero Urgente*** (16 bits): Indica el desplazamiento que es necesario añadir al Número de Secuencia del segmento para determinar el último octeto de datos urgentes que éste transporta. Para ser válido requiere que la bandera URG esté activada.

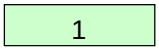
- ***Opciones*** (longitud variable): Campos opcionales. Las únicas opciones definidas en la especificación inicial de TCP son la **lista de final de opción**, **no operación** y la opción de **tamaño máximo de segmento**. No obstante, se definen nuevas opciones TCP entre las que se encuentran la opción de **escalado de ventana** [RFC1323], la de **marca temporal** o la de **descubrimiento de la MTU del enlace** [RFC1191]. Ocupa como máximo 40 octetos.

  El formato general es el siguiente:



  Algunas de las opciones definidas son las siguientes:

  • *Fin de lista de opciones*: Indica el final de la lista de opciones 

  • *Sin operación*: Delimitador de opciones. Hace coincidir cada opción con el inicio de una palabra de 32 bits 

  • *Tamaño máximo de segmento (MSS)*: Se negocia en la fase de establecimiento de la conexión. Si se utilizan paquetes demasiado pequeños existe ineficiencia en cuanto a la relación datos y cabecera. Si se utilizan paquetes demasiado grandes, será necesaria fragmentación, con lo que la pérdida de un fragmento implicará la retransmisión del paquete entero. Si no se conoce información de la red, lo óptimo será que el datagrama IP se adecue al tamaño de la MTU (*Maximum Transmission Unit)*, y el segmento TCP al del datagrama IP.

  En caso de redes con muchos errores deberá encontrarse un punto óptimo en cuanto a ineficiencia de cabeceras y probabilidad de pérdida de segmento.

  El formato es el siguiente:

• *Factor de escalado de ventana*: Transforma el valor del tamaño máximo de la ventana de transmisión de 16 a 32 bits. Es adecuado para redes con un producto retardo ancho de banda elevado. Si el Contador de desplazamiento vale 0, indica sin escalado. El escalado máximo es 14, lo que nos lleva a 65535*2^14.

3	3	Contador desplazamiento

1 byte

Esta opción solamente puede aparecer en un segmento SYN. Se fija en el establecimiento y se mantiene a lo largo de la conexión pudiendo ser diferente en los dos sentidos

• *Marca Temporal*: Permite al emisor enviar una referencia de tiempo en cada segmento. El receptor hace un eco del tiempo en el reconocimiento. De esta forma el emisor puede calcular el tiempo de ida y vuelta del segmento en cuestión.

8	10	Valor de timestamp	Eco timestamp

2.6 VERSIONES TCP

Finalmente, en este capítulo de TCP vamos a comentar algunos aspectos importantes de diferentes versiones del protocolo asociando a cada una de ellas el nombre por el cuál son conocidas.

- **TCP Tahoe**

La versión *TCP Tahoe* se incorporó al sistema operativo BSD en 1988. Fue la primera en incorporar los mecanismos de control de congestión y de estimación de RTT propuestos en [Jac88]. El primero de los mecanismos que fue introducido fue el de **Inicio Lento**, y el segundo el de la **estimación de RTT** basado, además de la media, en medidas de la **varianza**. Otro mecanismos incorporado fue el de **Prevención de la Congestión**. Por último, se introdujo el mecanismo de **Retransmisión Rápida**. *TCP Tahoe* tiene por tanto los mecanismos básicos de congestión y recuperación de pérdidas, y es el más común en las implementaciones actuales. No obstante, el principal inconveniente que presenta es el del Inicio Lento, concretamente en enlaces de retardo elevado, provocando el bajo rendimiento del protocolo.

- **TCP Reno**

La versión *TCP Reno* se incorporó al sistema operativo BSD en 1990. Éste incorpora todas las características de *TCP Tahoe* y añade el algoritmo de Recuperación Rápida que actúa conjuntamente con el de **Retransmisión Rápida**. De esta forma, tras la retransmisión no se

invoca el algoritmo de Inicio lento, sino el de Prevención de la Congestión, permitiendo una recuperación más rápida tras la retransmisión. El inconveniente más destacado es que, en caso de tener múltiples pérdidas por ventana, el protocolo de retransmisión rápida no puede recuperar de forma rápida más que la primera pérdida.

- **TCP New-Reno**

La versión *TCP New-Reno* se propuso en [Hoe96]. Se propone **una modificación al algoritmo de Recuperación Rápida** de forma que en caso de que existan varias pérdidas por ventana se soluciona el problema de *TCP Reno.*

- **TCP Vegas**

En 1995 se propone [BrP95a, BrP95b] otra versión del protocolo denominada *TCP Vegas*. En ella se modifican algunos aspectos de los algoritmos de Retransmisión y Recuperación Rápida, así y como del de Inicio Lento. Como aspecto más relevante, no obstante, es la propuesta a actuar contra la congestión antes de que ésta se detecte por la expiración del temporizador de retransmisión. *TCP Vegas* introduce un algoritmo para la predicción de la cantidad de datos que el enlace puede cursar sin congestión, e inyecta en el enlace dicha cantidad. Esta predicción se basa en medidas de caudal.