5 Compresión de Cabeceras de Van Jacobson

5.1 INTRODUCCIÓN

El acceso a servicios de Internet a través de líneas de baja velocidad tanto alámbricas como inalámbricas pone de manifiesto el hecho de la gran cantidad de información no útil que se envía en cada paquete TCP/IP, traduciéndose en una degradación de las prestaciones del protocolo. Además, ya hemos comentado que una de las características fundamentales de los enlaces radio móviles es el escaso ancho de banda del que disponen. Cualquier método destinado a incrementar la eficiencia en el uso de este recurso tan limitado puede resultar extremadamente interesante.

Uno de estos métodos consiste en **comprimir el tamaño de las cabeceras** de los protocolos de red para conseguir una mayor proporción de información útil transmitida frente a información de canal.

En primer lugar, constataremos las ventajas que presenta, a priori, esta estrategia de compresión y, en especial, nos centraremos en aquellas que atañen en mayor medida a los enlaces radio móviles. Dado que en este estudio se analiza el protocolo de comunicaciones TCP, a continuación, expondremos el funcionamiento concreto del algoritmo de compresión de cabeceras más extendido, el ideado por Van Jacobson descrito en el RFC1144, para reducir el tamaño de las cabeceras TCP/IP en enlaces con un ancho de banda reducido.

5.2 VENTAJAS DE LA COMPRESIÓN DE CABECERAS

La reducción del tamaño de las cabeceras de los paquetes que circulan por el enlace tiene numerosos aspectos positivos:

1. En primer lugar, permite mejorar el tiempo de respuesta percibido por un usuario de aplicaciones interactivas.

Pensemos en una de estas aplicaciones: *rlogin*. Cada uno de los caracteres que tecleamos al usar esta aplicación, se transmite en un paquete independiente. Esto significa que, en un enlace a 9600 bps y considerando el encapsulado TCP/IP/PPP medio (55 octetos), el eco de un simple carácter tardará aproximadamente 117ms, frente a los 100 ms considerados como límite del tiempo de respuesta interactiva aceptable [RFC1144] Utilizando la compresión de cabeceras propuesta por Van Jacobson, el tamaño de la cabecera se reduce a 3 ó 5 octetos por lo que este intervalo de respuesta podría descender hasta unos 8 ms.

2. La compresión permite utilizar paquetes pequeños (recomendable para el tráfico interactivo) sin menoscabo de la eficiencia en el uso del ancho de banda del canal (prioritaria en el tráfico masivo de datos).

Cuando se utilizan simultáneamente aplicaciones interactivas, como *telnet* o *rlogin*, con otras de transmisión masiva de datos, como *ftp*, se plantea un claro compromiso relacionado con el tamaño de paquete utilizado.

Por un lado, en la transmisión masiva de datos el factor que se intenta primar es el caudal, es decir, la utilización del ancho de banda del enlace. Para ello conviene utilizar paquetes de gran tamaño que minimicen la proporción de información de control (cabeceras) frente a información útil (datos del usuario) tal que se consuma poco ancho de banda en la transmisión del primer tipo de información.

No obstante, en esta situación de concurrencia de aplicaciones interactivas y de transmisión masiva de datos, cada vez que se escribe un carácter con *telnet* o *rlogin*, éste debe esperar, en media, para ser enviado, la mitad del tiempo de transmisión de un paquete de tamaño máximo por lo que, cuanto mayor sea éste, más negativa será la percepción del usuario de la respuesta a sus órdenes interactivas.

La compresión de cabeceras resulta, entonces, muy útil ya que contribuye a independizar, en gran medida, el tamaño de segmento del caudal resultante mediante la reducción del tamaño de las cabeceras a valores despreciables frente a la cantidad de información del usuario.

3. En particular, la compresión de cabeceras a priori parece ser especialmente aconsejable en los entornos móviles celulares debido a los valores muy reducidos que presenta la unidad máxima de transferencia ó *Maximum Transmission Unit* (MTU) de estos enlaces. Esta característica responde a la necesidad de minimizar la probabilidad de error de los paquetes que circulan por la red. Cuanto mayor es el tamaño de un paquete, mayor es su vulnerabilidad al carácter errático del canal móvil (recordemos que un error en uno sólo de los bits que constituyen una trama PPP se traduce, indefectiblemente, en la pérdida del paquete completo y, en consecuencia, en su ulterior retransmisión).

La compresión de cabeceras, por tanto, parece doblemente positiva ya que contribuye a incrementar la eficiencia en la utilización del escaso ancho de banda del canal radio a la vez que reduce la tasa de pérdida de paquetes comprimiendo su tamaño.

En la Figura 5.1 se compara, en función de la tasa de error en bit, la tasa de pérdida de paquetes en un enlace en el que la MTU vale 296 octetos para los casos en que se usa la compresión de cabeceras diseñada por Van Jacobson y en que no se utiliza ningún tipo de compresión.

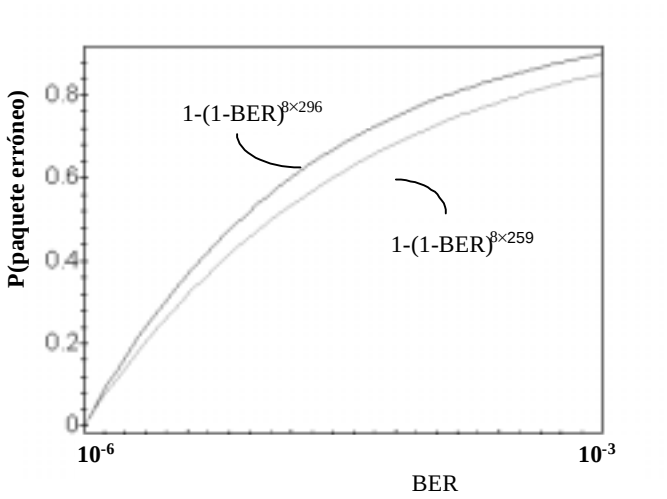

Figura 5.1 La compresión de cabeceras reduce la probabilidad de paquete erróneo para una tasa de error en bit dada

Cabe tener en cuenta, además, que para IPv6 el tamaño de las cabeceras TCP/IP es mucho mayor que el típico de 40 octetos. Frente a los 4 octetos utilizados para codificar las direcciones en la versión 4 del protocolo IP (IPv4), por ejemplo, la nueva versión del protocolo (IPv6) propone 16, de manera que la cabecera TCP/IPv6 pasa a ocupar 60 octetos. Además, con el objetivo de permitir la movilidad de los terminales, el *datagrama* IP se modifica para incluir nuevos campos que ayudan al enrutado del terminal móvil. Esta nueva funcionalidad supone un incremento en el tamaño de la cabecera TCP/IPv4 de 20 octetos (lo que resulta en un total de 60 octetos) y de 40 octetos en el caso de la cabecera TCP/IPv6 (que pasa a ocupar, en total, 100 octetos). Por tanto, la compresión de cabeceras parece poder llegar a ser imprescindible, en entornos como el móvil.

5.3 EL ALGORITMO VAN JACOBSON DE COMPRESIÓN DE CABECERAS TCP/IP

El algoritmo de compresión de cabeceras TCP/IP diseñado por Van Jacobson y descrito en el RFC 1144 permite reducir el tamaño de las cabeceras TCP/IP (típicamente 40) hasta 3 ó 5 octetos dependiendo de la aplicación que vaya encapsulada en el *datagrama*. Este algoritmo, está pensado, únicamente, para cabeceras TCP/IP en las que la versión del protocolo IP es la 4.

5.3.1 FUNCIONAMIENTO GENERAL DEL ALGORITMO VAN JACOBSON

El algoritmo de compresión Van Jacobson se basa en la idea de que la mayoría de los campos de la cabecera TCP/IP o bien se mantienen fijos durante el tiempo que dura la conexión o bien varían muy poco.

En la Figura 5.2 se muestran, sin sombrear, los campos que sí cambian a lo largo de la conexión.

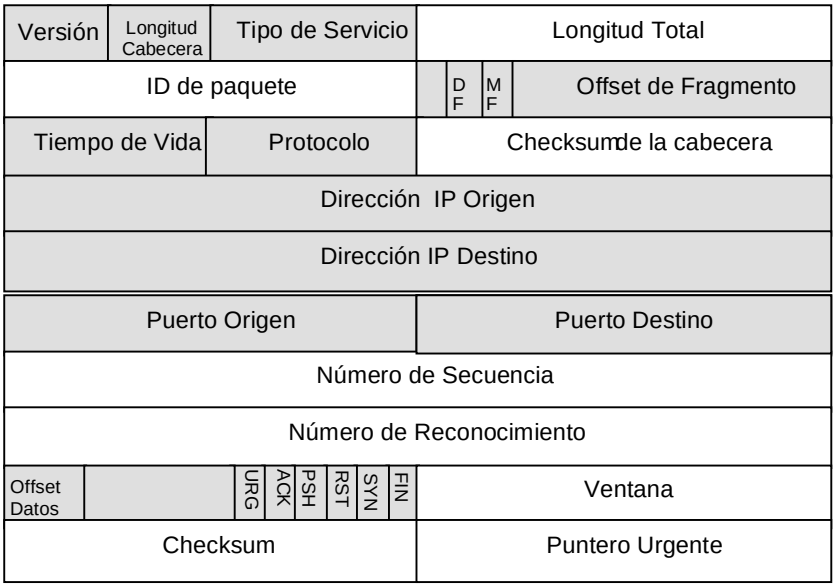

Figura 5.2 Los campos fijos de la cabecera TCP/IP se muestran sombreados

Un estudio detallado de la evolución de los diferentes campos permite constatar que 5 octetos de cabecera son suficientes para transmitir toda la información necesaria. En el caso de tráfico interactivo y transferencias masivas de datos, es posible llegar a niveles de compresión incluso mayores (cabeceras de 3 octetos).

Los campos que varían a lo largo de la conexión son sustituidos por **incrementos** respecto al valor del mismo campo del segmento inmediatamente anterior, de manera que, por ejemplo, el número de secuencia de cada segmento recibido se calcula a partir del segmento anterior sumándole el valor incremental indicado por el nuevo segmento.

El diagrama de bloques del modelo de compresión se muestra en la Figura 5.3.

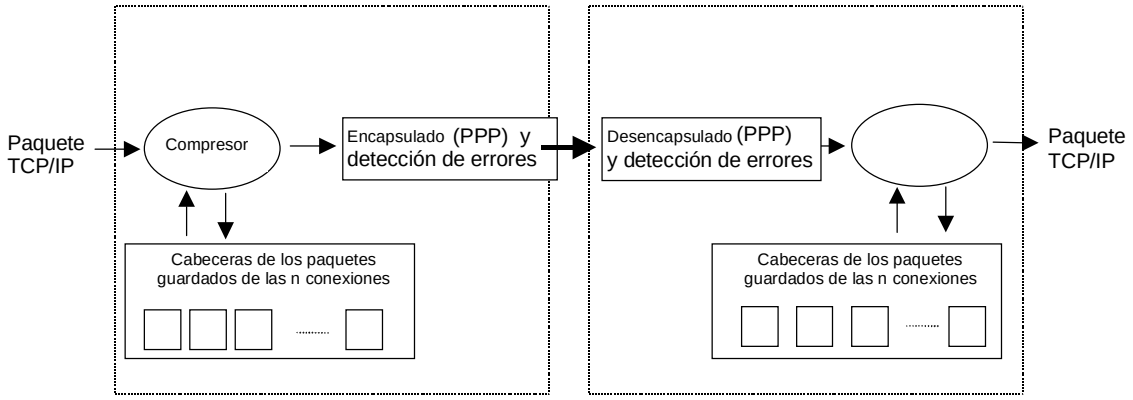

Figura 5.3 Modelo de compresión/descompresión

El nivel de red pasa el *datagrama* IP que debe ser enviado al nivel de enlace (PPP). El paquete atraviesa el bloque compresor que comprueba si el protocolo que va encapsulado en IP es TCP (el algoritmo sólo se aplica a cabeceras TCP/IP). Si el paquete es, efectivamente, TCP y debe ser comprimido, se lleva a cabo la compresión y se entrega el paquete comprimido al bloque encargado del encapsulado PPP el cual genera la trama correspondiente.

En recepción, el bloque encargado de desencapsular la trama PPP comprueba que el *Checksum* de la trama sea correcto. Si es así el paquete pasa al descompresor; en caso contrario es descartado directamente. El descompresor se encarga de descomprimir la cabecera TCP/IP y pasar el paquete resultante a la capa de red (IP).

En la Figura 5.4 se muestra el formato de la cabecera de un *datagrama* TCP/IP comprimido. Se utiliza una ***máscara de cambio*** que identifica aquellos campos de la cabecera cuyo valor ha variado respecto al paquete anterior (si el bit correspondiente vale 1 indica que el campo en cuestión está presente en el *datagrama* comprimido). También incluye un identificador de conexión de manera que el descompresor en recepción pueda localizar el último segmento recibido de esa misma conexión para, a partir de él, calcular el valor de los campos del segmento actual (téngase en cuenta que un mismo enlace puede ser compartido por varias conexiones). Los campos correspondientes al *Checksum* y el de puntero urgente no son comprimidos. El campo de Identificador IP solamente se transmite si es diferente de 1. Finalmente, para cada uno de los campos indicados por la máscara de cambio se incluye el valor concreto de la diferencia respecto al segmento anterior (valor incremental).

Los campos Puntero Urgente, ΔVentana, ΔReconocimiento y ΔSecuencia pueden ser de 1 o 3 octetos, dependiendo del valor que tienen que representar.

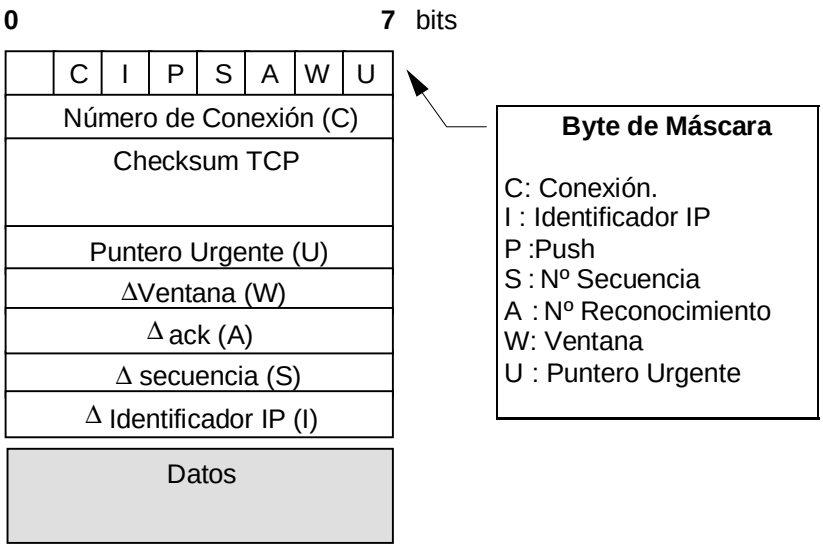

Figura 5.4 Cabecera de un datagrama IP comprimido

5.3.2 CASOS ESPECIALES

El algoritmo de compresión de cabeceras de Van Jacobson tiene algunos casos especiales, en los que no se sigue el esquema de compresión anteriormente descrito. La definición de éstos se describe en el RFC1144. El más importante a tener en cuenta en este estudio es el relativo a transferencias **unidireccionales**.

Al analizarse este tipo de transferencias mediante la aplicación *ftp*, podrá darse el caso de que de un segmento al siguiente, solamente cambie el número de secuencia. En este caso, el compresor debería mandar la cabecera comprimida con la máscara **0000 1000**, los dos octetos de *checksum* y el incremento en el número de secuencia. No obstante, si se da el caso que este incremento es igual a la cantidad de datos que había en el segmento anterior, no será necesario mandar este incremento. Para indicar al descompresor que se trata de este caso particular, se mandará una máscara **0000 11111** (0F hex ó }/). De esta forma se reduce la cabecera a 4 octetos (se añade uno debido al carácter de escape de *ppp*).

5.3.3 INTERACCIÓN CON LOS ERRORES

Debido a que los paquetes son reconstruidos por el descompresor a partir de los datos obtenidos del paquete anterior, **un error en uno de los paquetes se propaga y afecta a todos los posteriores a él**. Por esta razón, el algoritmo de compresión debe estar preparado para detectar posibles situaciones de error y recuperarse ante ellas.

En general, pueden distinguirse dos tipos de error diferentes: por un lado, los que provocan la recepción de un paquete corrupto, los cuales son detectados mediante el *Checksum* incorporado en la trama física (trama PPP), y, por otro, los que provocan la pérdida total del

paquete, es decir, hacen que éste no sea ni siquiera detectado en recepción (se perciba como ruido). Estos últimos no son detectados por el bloque descompresor pero sí por TCP al calcular el *Checksum* de la cabecera de los siguientes segmentos recibidos.

A continuación veremos cómo se protege el mecanismo de compresión de Van Jacobson de ambos tipos de error.

Ante el primer tipo de error expuesto anteriormente, el desencapsulador descarta la trama y envía al descompresor una señal para indicarle que debe desechar todos los paquetes hasta que reciba uno sin comprimir (es decir, con valores absolutos, y no relativos, del número de secuencia, número de reconocimiento, etc.).

Ante el segundo tipo de error, el TCP receptor descarta los paquetes recibidos tras el afectado por el error debido a su *Checksum* TCP erróneo y, por tanto, no envía ningún tipo de reconocimiento al emisor.

En ambos casos, pues, el receptor queda a la espera de que el emisor envíe un nuevo paquete no comprimido.

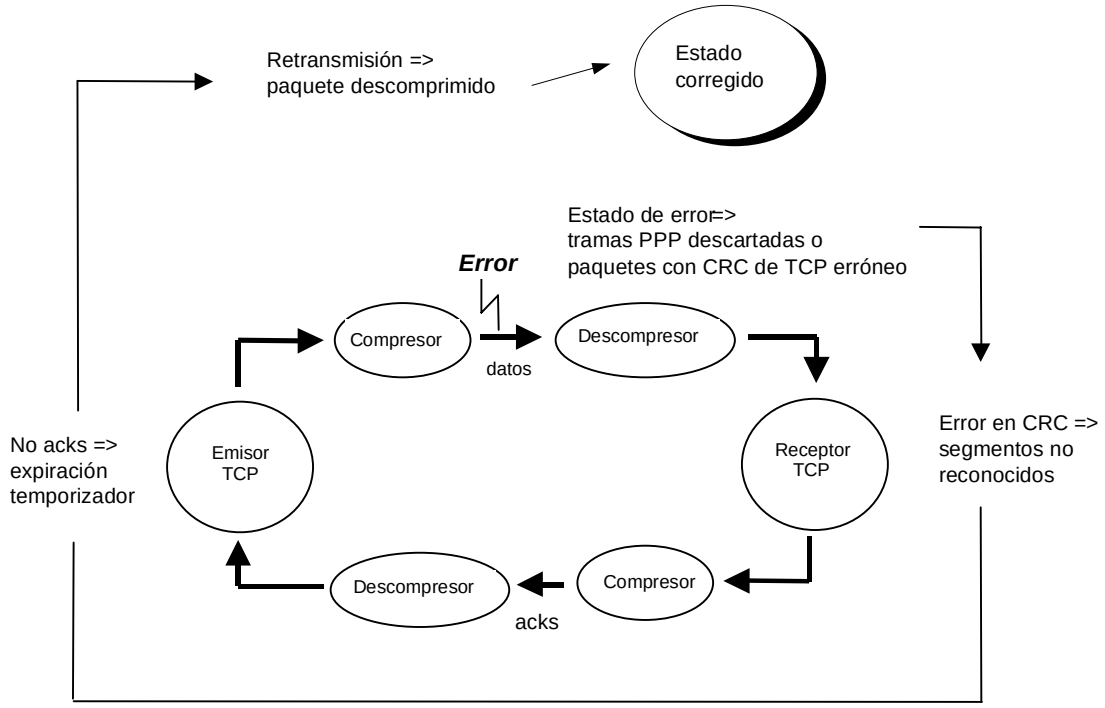

Figura 5.5 Proceso de recuperación ante errores que afectan a segmentos de datos

Puesto que el descompresor en recepción no envía ningún tipo de señal al compresor en emisión, todo queda en manos del TCP emisor que, al detectar la expiración del temporizador de retransmisión del segmento erróneo, retransmite este segmento. El compresor en emisión detecta, entonces, que el número de secuencia del nuevo paquete es menor o igual que el del último enviado y es obligado a transmitir el paquete sin comprimir de manera que el receptor pueda volver a sincronizarse y todo vuelva a su cauce normal.

Este mecanismo de detección y recuperación de errores en segmentos se muestra de manera esquemática en la Figura 5.5.

Por supuesto, el sistema también permite la recuperación cuando se producen errores que afectan a reconocimientos.

Si el error se detecta en el CRC de la trama física, el desencapsulador avisa al descompresor para que descarte cualquier paquete posterior hasta la llegada de uno descomprimido.

Si el error es debido a la pérdida del paquete de reconocimiento, se detecta en el *Checksum* de la cabecera TCP de los reconocimientos posteriores, los cuales son inmediatamente descartados. Esto no quiere decir que el carácter acumulativo de los reconocimientos deje de ser útil en este caso.

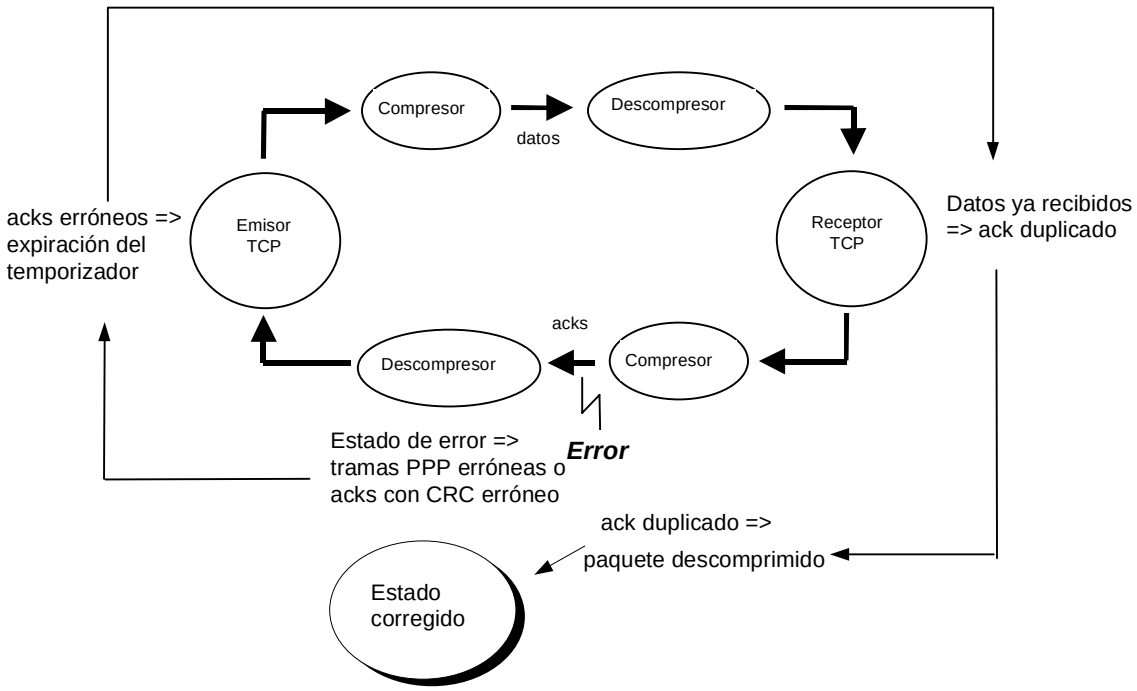

Figura 5.6 Proceso de recuperación ante errores que afectan a reconocimientos

De nuevo todo vuelve a la normalidad cuando, ante la expiración del temporizador del segmento cuyo reconocimiento se ha perdido o corrompido, el emisor lo retransmite y el compresor es forzado a enviar el segmento sin comprimir. Cuando el TCP en recepción detecta este paquete duplicado (puesto que, de hecho, ya lo había recibido anteriormente) envía un reconocimiento idéntico al anterior, es decir, un reconocimiento duplicado. Cuando el bloque compresor en recepción lo detecta, es obligado a enviar un paquete no comprimido. El esquema, para este caso, se muestra en la Figura 5.6.

En este estudio veremos si realmente es adecuado o no aplicar este tipo de algoritmos en el entorno móvil, qué inconvenientes se plantean y qué mejoras se proponen.