(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 399 220**

(51) Int. CI.:

H03M 13/11 (2006.01)
H04L 1/00 (2006.01)
H03M 13/37 (2006.01)
H04L 12/18 (2006.01)

(12) TRADUCCIÓN DE PATENTE EUROPEA T3

(96) Fecha de presentación y número de la solicitud europea: **17.09.1999 E 10011741 (5)**

(97) Fecha y número de publicación de la concesión europea: **07.11.2012 EP 2290826**

(54) Título: **Procedimiento de recuperación de paquetes perdidos para protocolos de transmisión de paquetes**

(30) Prioridad:

23.09.1998 US 101473 P
05.02.1999 US 246015

(45) Fecha de publicación y mención en BOPI de la traducción de la patente:
26.03.2013

(73) Titular/es:

DIGITAL FOUNTAIN, INC. (100.0%)
5775 Morehouse Drive
San Diego, CA 92121, US

(72) Inventor/es:

LUBY, MICHAEL G.

(74) Agente/Representante:

CARPINTERO LÓPEZ, Mario

DESCRIPCIÓN

Procedimiento de recuperación de paquetes perdidos para protocolos de transmisión de paquetes

Antecedentes de la invención

La presente invención se refiere a la codificación y descodificación de datos en sistemas de comunicación y, más específicamente, a sistemas de comunicación que codifican y descodifican datos para tener en cuenta errores y brechas en los datos comunicados, y para utilizar eficazmente los datos comunicados que emanan desde más de un origen.

La transmisión de ficheros entre un remitente y un destinatario por un canal de comunicaciones ha sido tema de mucha bibliografía. Preferiblemente, un destinatario desea recibir una copia exacta de los datos transmitidos por un canal por un remitente, con cierto nivel de certeza. Allí donde el canal no tiene fidelidad perfecta (lo que abarca la mayoría de todos los sistemas físicamente realizables), una preocupación es cómo tratar los datos perdidos o desfigurados en la transmisión. Los datos perdidos (borrados) son a menudo más fáciles de tratar que los datos desfigurados (errores), porque el destinatario no siempre puede decir cuándo los datos desfigurados son datos recibidos con errores. Muchos códigos de corrección de errores han sido desarrollados para corregir borrados (los llamados “códigos de borrado”) y / o errores (“códigos correctores de errores” o “ECC”). Habitualmente, el código específico usado se escoge en base a alguna información acerca de las infidelidades del canal a través del cual se están transmitiendo los datos y la naturaleza de los datos que se están transmitiendo. Por ejemplo, allí donde se conoce que el canal tiene largos periodos de infidelidad, un código de errores de ráfaga podría ser el más adecuado para esa aplicación. Allí donde solamente se esperan errores breves e infrecuentes, un código de paridad sencilla podría ser lo mejor.

La transmisión de ficheros entre múltiples remitentes y / o múltiples receptores por un canal de comunicaciones también ha sido tema de mucha bibliografía. Habitualmente, la transmisión de ficheros desde múltiples remitentes requiere la coordinación entre los múltiples remitentes para permitir que los remitentes minimicen la duplicación de esfuerzos. En un típico sistema de múltiples remitentes que envían un fichero a un receptor, si los remitentes no coordinan qué datos transmitirán y cuándo, sino que, en cambio, simplemente envían segmentos del fichero, es probable que un receptor reciba muchos segmentos duplicados inservibles. De manera similar, allí donde distintos receptores se incorporan a una transmisión desde un remitente en distintos momentos en el tiempo, una preocupación es cómo asegurar que todos los datos que reciben los receptores desde el remitente sean útiles. Por ejemplo, supongamos que el remitente está transmitiendo continuamente datos acerca del mismo fichero. Si el remitente solamente envía segmentos del fichero original y se pierden algunos segmentos, es probable que un receptor reciba muchos segmentos duplicados inservibles antes de recibir una copia de cada segmento en el fichero.

Otra consideración al seleccionar un código es el protocolo usado para la transmisión. En el caso de la intrared global de redes conocida como “Internet” (con “I” mayúscula), se usa un protocolo de paquetes para el transporte de datos. Ese protocolo se llama el Protocolo de Internet o “IP”, para abreviar. Cuando un fichero, u otro bloque de datos, ha de ser transmitido por una red del IP, se divide en símbolos de entrada de igual tamaño y los símbolos de entrada se colocan en paquetes consecutivos. Al estar basado en paquetes, un esquema de codificación orientado a paquetes podría ser adecuado. El “tamaño” de un símbolo de entrada puede ser medido en bits, ya sea que el símbolo de entrada esté o no efectivamente dividido en un flujo de bits, donde un símbolo de entrada tiene un tamaño de M bits cuando el símbolo de entrada se selecciona entre un alfabeto de 2^M símbolos.

El Protocolo de Control de Transporte (“TCP”) es un esquema de control de paquetes punto a punto de uso común, que tiene un mecanismo de acuse de recibo. El TCP es bueno para las comunicaciones de uno a uno, donde tanto el remitente como el destinatario acuerdan cuándo tendrá lugar y se recibirá la transmisión, y ambos acuerdan cuáles transmisores y receptores se usarán. Sin embargo, el TCP a menudo no es adecuado para comunicaciones de uno a muchos o de muchos a muchos, o allí donde el remitente y el destinatario determinan independientemente cuándo y dónde transmitirán o recibirán datos.

Al usar el TCP, un remitente transmite paquetes ordenados y el destinatario acusa recibo de cada paquete. Si se pierde un paquete, no se enviará ningún acuse de recibo al remitente y el remitente reenviará el paquete. La pérdida de paquetes tiene un cierto número de causas. En Internet, la pérdida de paquetes a menudo ocurre porque la congestión esporádica causa que el mecanismo de almacenamiento temporal en un encaminador colme su capacidad, forzándolo a descartar paquetes entrantes. Con protocolos tales como TCP / IP, el paradigma de acuse de recibo permite que los paquetes se pierdan sin fallo total, dado que los paquetes perdidos pueden ser simplemente retransmitidos, bien en respuesta a una falta de acuse de recibo o bien en respuesta a una solicitud explícita del destinatario. En cualquier caso, un protocolo de acuse de recibo requiere un canal de retorno desde el destinatario al remitente.

Aunque los protocolos basados en acuse de recibo son generalmente adecuados para muchas aplicaciones y, de hecho, se usan extensamente por la Internet actual, son ineficaces y, a veces, completamente inviables para ciertas aplicaciones. En particular, los protocolos basados en acuse de recibo responden mediocremente en redes con altas latencias, altas tasas de pérdida de paquetes, altas y bajas no coordinadas de destinatarios y / o ancho de banda sumamente asimétrico.

La alta latencia se da allí donde los acuses de recibo se toman un largo tiempo para viajar desde el destinatario de regreso al remitente. La alta latencia puede dar como resultado que el tiempo global antes de una retransmisión sea prohibitivamente largo. Las altas tasas de pérdida de paquetes también causan problemas allí donde varias retransmisiones del mismo paquete pueden dejar de llegar, lo que conduce a un largo retardo para obtener el último, o los pocos últimos, paquetes sin suerte.

Las "altas y bajas no coordinadas de destinatarios" se refieren a la situación donde cada destinatario puede incorporarse a, y abandonar, una sesión de transmisión en marcha, según su propio criterio. Esta situación es típica en Internet, los servicios de próxima generación tales como el "vídeo a petición" y otros servicios a ser ofrecidos por los proveedores de red en el futuro. En el típico sistema, si un destinatario se incorpora a, y abandona, una transmisión en marcha sin la coordinación de los remitentes, el destinatario probablemente percibirá una pérdida de un gran número de paquetes, con patrones de pérdidas ampliamente dispares percibidos por los distintos destinatarios.

El ancho de banda asimétrico se refiere a la situación donde un trayecto inverso de datos desde el destinatario al remitente (el canal de retorno) está menos disponible, o es más costoso, que el trayecto directo. El ancho de banda asimétrico puede hacer que sea prohibitivamente lento y / o caro para el destinatario acusar recibo de paquetes frecuentemente y los acuses infrecuentes de recibo pueden nuevamente introducir retardos.

Además, los protocolos basados en acuses de recibo no se adaptan bien para la difusión, donde un remitente está enviando simultáneamente un fichero a múltiples usuarios. Por ejemplo, supongamos que un remitente está difundiendo un fichero a múltiples destinatarios por un canal satelital. Cada destinatario puede experimentar un patrón distinto de pérdida de paquetes. Los protocolos que dependen de datos de acuse de recibo (ya sea positivo o negativo) para el suministro fiable del fichero requieren un canal de retorno desde cada destinatario al remitente, y esto puede ser prohibitivamente caro de proporcionar. Además, esto requiere un remitente complejo y potente, para ser capaz de gestionar debidamente todos los datos de acuse de recibo enviados desde los destinatarios. Otro inconveniente es que si distintos destinatarios pierden distintos conjuntos de paquetes, la redifusión de paquetes perdidos solamente por unos pocos de los destinatarios causa la recepción de paquetes duplicados inservibles por parte de otros destinatarios. Otra situación que no se gestiona bien en un sistema de comunicación basado en acuses de recibo es aquella donde los destinatarios pueden comenzar una sesión de recepción asíncronamente, es decir, el destinatario podría comenzar a recibir datos en el medio de una sesión de transmisión.

Se han sugerido varios esquemas complejos para mejorar las prestaciones de los esquemas basados en acuses de recibo, tales como el TCP / IP para la multidifusión y la difusión. Sin embargo, ninguno de ellos ha sido claramente adoptado en este momento, por diversas razones. Por lo pronto, los protocolos basados en acuses de recibo tampoco se adaptan bien allí donde un destinatario está obteniendo información desde múltiples remitentes, tal como en una red de difusión satelital de órbita terrestre baja ("LEO"). En una red LEO, los satélites LEO pasan por encima rápidamente debido a su órbita, por lo que el destinatario está solamente a la vista de algún satélite específico durante un breve tiempo. Para compensar esto, la red LEO comprende muchos satélites y los destinatarios son traspasados entre los satélites según un satélite desciende bajo el horizonte y otro asciende. Si se usara un protocolo basado en acuses de recibo para garantizar la fiabilidad, probablemente se requeriría un complejo protocolo de traspaso para coordinar que los acuses de recibo vuelvan al satélite adecuado, ya que un destinatario a menudo estaría recibiendo un paquete desde un satélite y, sin embargo, estar acusando recibo de ese paquete a otro satélite.

Una alternativa para un protocolo basado en acuses de recibo que a veces se pone en práctica es un protocolo basado en un carrusel. Un protocolo de carrusel divide un fichero de entrada en símbolos de entrada de igual longitud, coloca cada símbolo de entrada en un paquete y luego recicla continuamente y transmite todos los paquetes. Un inconveniente mayor para un protocolo basado en un carrusel es que si un destinatario pierde incluso un paquete, entonces el destinatario tiene que esperar otro ciclo entero antes de tener una oportunidad de recibir el paquete perdido. Otra forma de ver esto es que un protocolo basado en un carrusel puede ocasionar una gran magnitud de recepción inservible de datos duplicados. Por ejemplo, si un destinatario recibe paquetes desde el principio del carrusel, detiene la recepción por un rato y luego comienza a recibir nuevamente al comienzo del carrusel, se recibe un gran número de paquetes duplicados inservibles.

Una solución que se ha propuesto para resolver los problemas anteriores es evitar el uso de un protocolo basado en acuses de recibo y usar en cambio códigos de borrado, tales como los Códigos de Reed-Solomon, para aumentar la fiabilidad. Una característica de varios códigos de borrado es que, cuando un fichero es segmentado en símbolos de entrada que son enviados en paquetes al destinatario, el destinatario puede descodificar los paquetes para reconstruir el fichero entero, una vez que se han recibido suficientes paquetes, independientemente, en general, de cuáles paquetes llegan. Esta propiedad elimina la necesidad de acuses de recibo al nivel de los paquetes, dado que el fichero puede ser recuperado incluso si se pierden paquetes. Sin embargo, muchas soluciones de códigos de borrado no logran resolver los problemas del protocolo basado en acuses de recibo, o bien suscitan nuevos problemas.

Un problema con muchos códigos de borrado es que requieren excesiva potencia informática, o de memoria, para funcionar. Un esquema de codificación que ha sido desarrollado recientemente para aplicaciones de comunicaciones que

es algo eficaz en su uso de la potencia informática y de la memoria es el esquema de codificación Tornado.

Ejemplos de códigos Tornado y de sus implementaciones se describen en el documento de Byers, J., et al., "A Digital Fountain Approach to Reliable Distribution of Bulk Data" ["Un enfoque de fuente digital para la distribución fiable de datos a granel"], Informe Técnico ICSI Nº TR-98-005 (1998) y en los Anales de la Conferencia ACM SIGCOMM '98, Vol. 28, nº 4, págs. 56-67 (1998).

Los códigos Tornado son similares a los códigos de Reed-Solomon en cuanto a que un fichero de entrada está representado por K símbolos de entrada y se usa para determinar N símbolos de salida, donde N está fijado antes de que comience el proceso de codificación. La codificación con códigos Tornado es generalmente mucho más rápida que la codificación con códigos de Reed-Solomon, ya que el número medio de operaciones aritméticas requeridas para crear los N símbolos Tornado de salida es proporcional a N (del orden de decenas de operaciones de código ensamblador multiplicadas por N) y el número total de operaciones aritméticas requeridas para descodificar el fichero entero también es proporcional a N:

Los códigos Tornado tienen ventajas, en cuanto a velocidad, con respecto a los códigos de Reed-Solomon, pero con varias desventajas. En primer lugar, el número de símbolos de salida, N, debe estar determinado con anterioridad al proceso de codificación. Esto lleva a ineficiencias si la tasa de pérdidas de paquetes está sobre-estimada, y puede conducir al fracaso si la tasa de pérdidas de los paquetes está subestimada. Esto es porque un descodificador Tornado requiere un cierto número de símbolos de salida (específicamente, K + A paquetes, donde A es pequeño en comparación con K) para descodificar y restaurar el fichero original y, si el número de paquetes perdidos es mayor que N – (K + A), entonces el fichero original no puede ser restaurado. Esta limitación es generalmente aceptable para muchos problemas de comunicaciones, mientras N sea seleccionado mayor que K + A, en al menos la pérdida efectiva de paquetes, pero esto requiere una adivinación anticipada de la pérdida de paquetes.

Otra desventaja de los códigos Tornado es que requieren que el codificador y el descodificador concuerden, de alguna manera, en una estructura de gráfico. Los códigos Tornado requieren una etapa de pre-procesamiento en el descodificador donde se construye este gráfico, un proceso que frena sustancialmente la descodificación. Además, un gráfico es específico para un tamaño de fichero, por lo que se necesita generar un nuevo gráfico para cada tamaño de fichero usado. Además, los gráficos que necesitan los códigos Tornado son complicados para construir, y requieren distintas configuraciones personalizadas de parámetros para ficheros de distintos tamaños, para obtener las mejores prestaciones. Estos gráficos son de un tamaño significativo y requieren una cantidad significativa de memoria para su almacenamiento, tanto en el remitente como en el destinatario.

Además, los códigos Tornado generan exactamente los mismos valores de símbolos de salida con respecto a un gráfico y fichero de entrada fijados. Estos símbolos de salida consisten en los K símbolos de entrada originales y N-K símbolos redundantes. Además, N puede ser prácticamente sólo un pequeño múltiplo de K, tal como 1,5 o 2 veces K. Así, es muy probable que un destinatario que obtiene símbolos de salida generados a partir del mismo fichero de entrada, usando el mismo gráfico, desde más de un remitente, recibirá un gran número de símbolos de salida duplicados e inservibles. Esto es porque los N símbolos de salida están fijados de antemano y son los mismos N símbolos de salida que son transmitidos desde cada transmisor cada vez que los símbolos se envían, y son los mismos N símbolos recibidos por un receptor. Por ejemplo, supongamos que N = 1.500, K = 1.000 y que un receptor recibe 900 símbolos desde un satélite antes de que el satélite se sumerja por el horizonte. A menos que los satélites estén coordinados y en sincronización, los símbolos de código Tornado recibidos por el receptor desde el próximo satélite podrían no ser aditivos, porque el próximo satélite está transmitiendo los mismos N símbolos, lo que probablemente dé como resultado que el receptor reciba copias de muchos de los 900 símbolos ya recibidos antes de recibir 100 nuevos símbolos necesarios para recuperar el fichero de entrada.

Los códigos Tornado implican construcciones con gráficos bipartitos. Otros códigos que implican construcciones usando gráficos, específicamente, gráficos irregulares, se describen en el documento de Luby, M., et al., "Practical Loss-Resilient Codes" ["Códigos prácticos resistentes a pérdidas"], 29º Simposio Anual de ACM sobre la Teoría de la Computación, págs. 150-159 (1997). Otros códigos adicionales, utilizables para recuperar borrados con propiedades similares, se proporcionan en el documento de Alon, N., et al., "Linear Time Erasure Codes with Nearly Optimal Recovery" ["Códigos de borrado temporal lineal con recuperación casi óptima"], Proc. del 36º Simposio Anual sobre Fundamentos de la Ciencia de la Computación, págs. 512-19 (1995). Tales códigos tienen una específica tasa de código, que está habitualmente determinada con anterioridad a la transmisión.

Por lo tanto, lo que se necesita es un código sencillo de borrado que no requiera excesiva potencia informática o memoria en un remitente o destinatario para implementarse, y que pueda ser usado para distribuir eficazmente un fichero en un sistema con uno o más remitentes y / o uno o más destinatarios, sin requerir necesariamente la coordinación entre remitentes y destinatarios.

Resumen de la invención

De acuerdo a la presente invención, se proporciona un procedimiento y un aparato para transmitir datos desde un origen, a través de una pluralidad de canales, según lo estipulado en las reivindicaciones 1 y 10. Las realizaciones de la invención son reivindicadas en las reivindicaciones dependientes.

En un sistema de comunicaciones que usa la presente invención, un codificador usa un fichero de entrada de datos y una clave para producir un símbolo de salida, en donde el fichero de entrada es una pluralidad ordenada de símbolos de entrada, seleccionado cada uno a partir de un alfabeto de entrada, la clave es seleccionada a partir de un alfabeto de claves y el símbolo de salida es seleccionado a partir de un alfabeto de salida. Un símbolo de salida con clave I es generado determinando un peso, W(I), para el símbolo de salida a generar, en donde los pesos W son enteros positivos que varían entre al menos dos valores sobre la pluralidad de claves, seleccionando los W(I) de los símbolos de entrada asociados al símbolo de salida según una función de 1, y generando el valor B(I) del símbolo de salida a partir de una función F(I) de valor predeterminado de los W(I) símbolos de entrada seleccionados. El codificador puede ser llamado una o más veces, cada vez con otra clave, y cada vez produce un símbolo de salida. Los símbolos de salida son generalmente independientes entre sí, y puede generarse un número ilimitado (sujeto a la resolución o a I), si es necesario.

En un descodificador del sistema de comunicaciones, los símbolos de salida recibidos por un destinatario son símbolos de salida transmitidos desde un remitente, que generó esos símbolos de salida en base a una codificación de un fichero de entrada. Debido a que los símbolos de salida pueden perderse en el camino, el descodificador funciona adecuadamente incluso cuando solamente recibe una parte arbitraria de los símbolos de salida transmitidos. El número de símbolos de salida necesarios para descodificar el fichero de entrada es igual a, o levemente mayor que, el número de símbolos de entrada que comprende el fichero, suponiendo que los símbolos de entrada y los símbolos de salida representan el mismo número de bits de datos.

En un proceso de descodificación usado conjuntamente con la presente invención, las siguientes etapas son llevadas a cabo para cada símbolo de salida recibido: 1) identificar la clave I del símbolo de salida recibido; 2) identificar el valor B(I) del símbolo de salida recibido para el símbolo de salida; 3) determinar el peso, W(I), del símbolo de salida, 4) determinar las posiciones para los W(I) símbolos de entrada asociados al símbolo de salida, y 5) almacenar B(I) en una tabla de símbolos de salida, junto con el peso W(I) y las posiciones de los símbolos de entrada asociados. El siguiente proceso se aplica luego repetidamente hasta que no haya más símbolos de salida sin usar de peso uno: 1) para cada símbolo de salida almacenado que tenga un peso de uno y no esté indicado como un símbolo de salida “consumido”, calcular la posición J del único símbolo de entrada restante no recuperado asociado al símbolo de salida en base a su clave I; 2) calcular el valor para el símbolo J de entrada a partir del símbolo de salida; 3) identificar los símbolos de salida en la tabla de símbolos de salida que tienen el símbolo J de entrada como asociado; 4) recalcular los valores B para los símbolos de salida identificados para que sean independientes del símbolo J de entrada; 5) decrementar en uno los pesos de estos símbolos de salida identificados; y 6) indicar el símbolo J de entrada como recuperado y el símbolo de salida con la clave I como consumido. Este proceso se repite hasta que el conjunto ordenado de símbolos de entrada esté recuperado, es decir, hasta que todo el fichero de entrada esté completamente recuperado.

Una ventaja de la presente invención es que no requiere que el destinatario comience a recibir en cualquier momento dado en la transmisión y no requiere que el remitente se detenga después de que esté generado un número fijado de símbolos de salida, ya que el remitente puede enviar un conjunto efectivamente ilimitado de símbolos de salida para cualquier fichero de entrada. En cambio, un destinatario puede comenzar la recepción cuando esté listo, y desde cualquier lugar posible, y puede perder paquetes en un patrón aleatorio, y aún tener una buena oportunidad de que la gran mayoría de los datos recibidos sean datos “aditivos en información”, es decir, datos que ayudan en el proceso de recuperación, en lugar de ser duplicados de información ya disponible. El hecho de que flujos de datos generados independientemente (a menudo, desvinculados aleatoriamente) estén codificados de manera aditiva en cuanto a información conduce a muchas ventajas, en cuanto a que permite la generación y recepción de múltiples orígenes, la robustez de borrados y altas y bajas no coordinadas.

Una comprensión adicional de la naturaleza y las ventajas de las invenciones reveladas en el presente documento puede ser realizada por referencia a las partes restantes de la memoria y los dibujos adjuntos.

Breve descripción de los dibujos

La Fig. 1 es un diagrama de bloques de un sistema de comunicaciones útil para entender la presente invención.

La Fig. 2 es un diagrama de bloques que muestra el codificador de la Fig. 1 en mayor detalle.

La Fig. 3 es una ilustración de cómo un símbolo de salida podría ser generado a partir de un conjunto de símbolos de entrada asociados.

La Fig. 4 es un diagrama de bloques de un descodificador básico, como podría ser usado en el sistema de comunicaciones mostrado en la Fig. 1.

La Fig. 5 es un diagrama de bloques de un descodificador alternativo.

La Fig. 6 es un diagrama de flujo de un proceso que podría ser usado por un descodificador, tal como el descodificador mostrado en la Fig. 5, para recuperar símbolos de entrada a partir de un conjunto de símbolos de salida.

La Fig. 7 es un diagrama de flujo del proceso que podría ser usado por un organizador de recepción, tal como el organizador de recepción mostrado en la Fig. 5, para organizar los símbolos de salida recibidos.

La Fig. 8(a) es un diagrama de flujo del proceso que podría ser usado por un procesador de recuperación, tal como el procesador de recuperación mostrado en la Fig. 5, para procesar los símbolos de salida recibidos.

Las Figs. 8(b)-(c) forman un diagrama de flujo de partes de una variación del proceso de la Fig. 8(a), mostrando la Fig. 8(b) etapas llevadas a cabo en el proceso de recuperación, incluso el procesamiento diferido, y mostrando la Fig. 8(c) el procesamiento diferido.

La Fig. 9 es un diagrama de bloques que muestra al asociador de la Fig. 2 en mayor detalle.

La Fig. 10 es un diagrama de flujo de un proceso que podría ser usado por un asociador, tal como el asociador mostrado en la Fig. 9, para determinar rápidamente la asociación de símbolos de entrada con símbolos de salida.

La Fig. 11 es un diagrama de bloques que muestra el selector de pesos de la Fig. 2 en mayor detalle.

La Fig. 12 es un diagrama de flujo de un proceso que podría ser un usado por un selector de pesos, tal como el selector de pesos mostrado en la Fig. 11, para determinar un peso para un símbolo de salida dado.

La Fig. 13 es un diagrama de flujo de un proceso para descodificar, para un descodificador que no necesita ser especialmente eficaz.

La Fig. 14 es un diagrama de bloques de un descodificador más eficaz.

La Fig. 15 es un diagrama de flujo de un proceso para descodificar, como podría ser implementado usando el descodificador de la Fig. 14 para descodificar más eficazmente que la descodificación descrita con referencia a las Figs. 12 a 13.

La Fig. 16 es un diagrama que ilustra un ejemplo de un documento y de símbolos de salida recibidos para el proceso de descodificación de la Fig. 15.

La Fig. 17 ilustra el contenido de tablas en un descodificador durante el proceso de descodificación mostrado en la Fig. 15.

La Fig. 18 es un diagrama que ilustra el contenido de una tabla de clasificación de pesos, como podría ser usada durante el proceso de descodificación mostrado en la Fig. 15.

La Fig. 19 ilustra una lista de ejecución que podría ser formada durante el proceso de descodificación mostrado en la Fig. 15.

La Fig. 20 ilustra el avance del proceso de recuperación en forma de un gráfico del tamaño de conjunto descodificable con respecto al número de símbolos de entrada recuperados para una distribución ideal.

La Fig. 21 ilustra el avance del proceso de recuperación en forma de un gráfico del tamaño del conjunto descodificable con respecto al número de símbolos de entrada recuperados para una distribución robusta de pesos.

La Fig. 22 es una ilustración de un sistema de comunicación punto a punto entre un remitente (transmisor) y un receptor, usando un codificador y un descodificador, según lo ilustrado en las figuras anteriores.

La Fig. 23 es una ilustración de un sistema de comunicación por difusión entre un remitente y múltiples receptores (solamente uno de los cuales se muestra) que usa un codificador y un descodificador, según lo ilustrado en las figuras anteriores.

La Fig. 24 es una ilustración de un sistema de comunicaciones según una realización de la presente invención, donde un receptor recibe símbolos de salida desde múltiples remitentes, usualmente independientes.

La Fig. 25 es una ilustración de un sistema de comunicación según una realización de la presente invención, donde múltiples receptores, posiblemente independientes, reciben símbolos de salida desde múltiples remitentes, usualmente

independientes, para recibir un fichero de entrada en menos tiempo que si usara solamente un receptor y / o solamente un remitente.

El Apéndice A es un listado del código fuente de un programa para implementar una distribución de pesos.

Descripción de las realizaciones preferidas

En los ejemplos descritos en el presente documento, se describe un esquema de codificación indicado como “codificación por reacción en cadena”, precedido por una explicación del significado y alcance de los diversos términos usados en esta descripción.

Con la codificación por reacción en cadena, los símbolos de salida son generados por el remitente desde el fichero de entrada, según se necesiten. Cada símbolo de salida puede ser generado sin importar cómo son generados otros símbolos de salida. En cualquier instante en el tiempo, el remitente puede dejar de generar símbolos de salida y no es necesario que haya ninguna restricción en cuanto a cuándo el remitente detiene o reanuda la generación de símbolos de salida. Una vez generados, estos símbolos pueden ser luego colocados en paquetes y transmitidos a su destino, conteniendo cada paquete uno o más símbolos de salida.

Según se usa en el presente documento, el término “fichero” se refiere a cualquier dato que esté almacenado en uno o más orígenes y que haya de ser entregado como una unidad a uno o más destinos. Así un documento, una imagen y un fichero de un servidor de ficheros, o dispositivo de almacenamiento de ordenador, son todos ejemplos de “ficheros” que pueden ser entregados. Los ficheros pueden ser de tamaño conocido (tal como una imagen de un megaocteto almacenado en un disco rígido) o bien pueden ser de tamaño desconocido (tal como un fichero tomado de la salida de una fuente de transmisión por flujo). En cualquier caso, el fichero es una secuencia de símbolos de entrada, donde cada símbolo de entrada tiene una posición en el fichero y un valor.

La transmisión es el proceso de transmitir datos desde uno o más remitentes a uno o más destinatarios, a través de un canal, a fin de entregar un fichero. Si un remitente está conectado con cualquier número de destinatarios por un canal perfecto, los datos recibidos pueden ser una copia exacta del fichero de entrada, ya que todos los datos serán recibidos correctamente. Aquí, suponemos que el canal no es perfecto, que es el caso para la mayoría de los canales del mundo real, o bien suponemos que los datos emanan de más de un remitente, que es el caso para algunos sistemas. De las muchas imperfecciones del canal, dos imperfecciones de interés son el borrado de datos y la incompletitud de los datos (que puede ser tratada como un caso especial del borrado de datos). El borrado de datos ocurre cuando el canal pierde o descarta datos. La incompletitud de los datos ocurre cuando un destinatario no comienza a recibir datos hasta que algunos de los datos ya lo han pasado, el destinatario deja de recibir datos antes de que termine la transmisión, o bien el destinatario, intermitentemente, se detiene y comienza nuevamente a recibir datos. Como un ejemplo de la incompletitud de datos, un remitente satelital en movimiento podría estar transmitiendo datos que representan un fichero de entrada, e iniciar la transmisión antes de que un destinatario esté a su alcance. Una vez que el destinatario está al alcance, los datos pueden ser recibidos hasta que el satélite sale del alcance, momento en el cual el destinatario puede redirigir su placa satelital (tiempo durante el cual no está recibiendo datos) para comenzar a recibir los datos acerca del mismo fichero de entrada que está siendo transmitido por otro satélite que se ha puesto dentro del alcance. Como debería ser evidente de la lectura de esta descripción, la incompletitud de datos es un caso especial del borrado de datos, ya que el destinatario puede tratar la incompletitud de datos (y el destinatario tiene los mismos problemas) como si el destinatario estuviese al alcance todo el tiempo, pero el canal perdió todos los datos, hasta el momento en que el destinatario comenzó a recibir datos. Además, como es bien conocido en el diseño de sistemas de comunicación, los errores detectables pueden ser el equivalente de borrados, descartando sencillamente todos los bloques de datos o símbolos que tienen errores detectables.

En algunos sistemas de comunicación, un destinatario recibe datos generados por múltiples remitentes, o por un remitente que usa múltiples conexiones. Por ejemplo, para acelerar una descarga, un destinatario podría conectarse simultáneamente con más de un remitente, para transmitir datos concernientes al mismo fichero. Como otro ejemplo, en una transmisión de multidifusión, múltiples flujos de datos de multidifusión podrían ser transmitidos para permitir a los destinatarios conectarse con uno o más de estos flujos, para correlacionar la velocidad de transmisión conjunta con el ancho de banda del canal que los conecta con el remitente. En todos dichos casos, una preocupación es asegurar que todos los datos transmitidos sean de uso independiente para un destinatario, es decir, que los datos de múltiples orígenes no sean redundantes entre los flujos, incluso cuando las velocidades de transmisión sean sumamente distintas para los distintos flujos, y cuando hay patrones arbitrarios de pérdidas.

En general, la transmisión es el acto de mover datos desde un remitente a un destinatario por un canal que conecta al remitente y al destinatario. El canal podría ser un canal en tiempo real, donde el canal mueve datos desde el remitente al destinatario según el canal obtiene los datos, o bien el canal podría ser un canal de almacenamiento que almacena algunos de, o todos, los datos en su tránsito desde el remitente al destinatario. Un ejemplo de esto último es el almacenamiento en disco u otro dispositivo de almacenamiento. En ese ejemplo, un programa o dispositivo que genera datos puede ser considerado como el remitente, transmitiendo los datos a un dispositivo de almacenamiento. El

destinatario es el programa o dispositivo que lee los datos desde el dispositivo de almacenamiento. Los mecanismos que el remitente usa para poner los datos sobre el dispositivo de almacenamiento, el mismo dispositivo de almacenamiento y los mecanismos que el destinatario usa para obtener los datos desde el dispositivo de almacenamiento forman, colectivamente, el canal. Si hay una oportunidad de que esos mecanismos o el dispositivo de almacenamiento puedan perder datos, entonces esto sería tratado como un borrado de datos en el canal.

Cuando el remitente y el destinatario están separados por un canal de borrado de datos, es preferible no transmitir una copia exacta de un fichero de entrada, sino transmitir, en cambio, datos generados desde el fichero de entrada, que asiste en la recuperación de borrados. Un codificador es un circuito, dispositivo, módulo o segmento de código que acomete esa tarea. Una forma de ver la operación del codificador es que el codificador genera símbolos de salida a partir de símbolos de entrada, donde una secuencia de valores de símbolos de entrada representa el fichero de entrada. Cada símbolo de entrada tendría, así, una posición en el fichero de entrada, y un valor. Un descodificador es un circuito, dispositivo, módulo o segmento de código que reconstruye los símbolos de entrada a partir de los símbolos de salida recibidos por el destinatario.

La codificación por reacción en cadena no está limitada a ningún tipo específico de símbolos de entrada, pero el tipo del símbolo de entrada está a menudo dictado por la aplicación. Habitualmente, los valores para los símbolos de entrada se seleccionan entre un alfabeto de 2^M símbolos, para algún entero positivo M. En tales casos, un símbolo de entrada puede ser representado por una secuencia de M bits de datos provenientes del fichero de entrada. El valor de M se determina a menudo en base a los usos de la aplicación y al canal. Por ejemplo, para un canal de Internet basado en paquetes, un paquete con una carga útil de un tamaño de 1.024 octetos podría ser adecuado (un octeto son 8 bits). En este ejemplo, suponiendo que cada paquete contiene un símbolo de salida y 8 octetos de información auxiliar, un tamaño de símbolo de entrada de M = (1.024 -8) . 8, o de 8.128 bits, sería adecuado. Como otro ejemplo, algunos sistemas satelitales usan el estándar de paquetes MPEG, donde la carga útil de cada paquete comprende 188 octetos. En ese ejemplo, suponiendo que cada paquete contenga un símbolo de salida y 4 octetos de información auxiliar, un tamaño de símbolo de M = (188 – 4) . 8, o de 1.472 bits, sería adecuado. En un sistema de comunicación de propósito general que usa la codificación por reacción en cadena, los parámetros específicos de la aplicación, tales como el tamaño de símbolos de entrada (es decir, M, el número de bits codificados por un símbolo de entrada), podrían ser variables fijadas por la aplicación.

Cada símbolo de salida tiene un valor. En una realización preferida, que consideramos más adelante, cada símbolo de salida tiene un identificador llamado su “clave”. Preferiblemente, la clave de cada símbolo de salida puede ser fácilmente determinada por el destinatario para permitir al destinatario distinguir un símbolo de salida de otros símbolos de salida. Preferiblemente, la clave de un símbolo de salida es distinta a las claves de todos los otros símbolos de salida. También preferiblemente, se incluyen tan pocos datos como sea posible en la transmisión a fin de que un destinatario determine la clave de un símbolo de salida recibido.

En una forma simple de formación de claves, la secuencia de claves para símbolos de salida consecutivos, generados por un codificador, es una secuencia de enteros consecutivos. En este caso, cada clave se llama un “número de secuencia”. En el caso en que hay un valor de símbolo de salida en cada paquete transmitido, el número de secuencia puede estar incluido en el paquete. Dado que los números de secuencia pueden caber habitualmente en un número pequeño de octetos, p. ej., 4 octetos, la inclusión del número de secuencia junto con el valor del símbolo de salida en algunos sistemas es económica. Por ejemplo, usando paquetes de Internet del UDP (Protocolo de Datos del Usuario) de 1.024 octetos cada uno, la adjudicación de 4 octetos en cada paquete para el número de secuencia incurre solamente en un pequeño sobregasto del 0,4%.

En otros sistemas, es preferible formar una clave a partir de más de un trozo de datos. Por ejemplo, consideremos un sistema que incluye un destinatario que recibe más de un flujo de datos generado desde el mismo fichero de entrada, proveniente de uno o más remitentes, donde los datos transmitidos son un flujo de paquetes, conteniendo cada uno un símbolo de salida. Si todos los flujos de ese tipo usan el mismo conjunto de números de secuencia como claves, entonces es probable que el destinatario reciba símbolos de salida con los mismos números de secuencia. Dado que los símbolos de salida con la misma clave o, en este caso, con el mismo número de secuencia, contienen idéntica información acerca del fichero de entrada, esto causa la recepción inútil de datos duplicados por parte del destinatario. De esta manera, en una situación de ese tipo se prefiere que la clave comprenda un único identificador de flujo asociado a un número de secuencia.

Por ejemplo, para un flujo de paquetes de Internet del UDP, el identificador único de un flujo de datos podría incluir la dirección de IP del remitente y el número de puerto que el remitente está usando para transmitir los paquetes. Dado que la dirección de IP del remitente y el número de puerto del flujo son partes de la cabecera de cada paquete del UDP, no hay ningún espacio adicional requerido en cada paquete para garantizar que estas partes de la clave estén disponibles para un destinatario. El remitente solamente necesita insertar un número de secuencia en cada paquete junto con el correspondiente símbolo de salida, y el destinatario puede recrear la clave entera de un símbolo de salida recibido a partir del número de secuencia y de la cabecera del paquete. Como otro ejemplo, para un flujo de paquetes de multidifusión de IP, el identificador único de un flujo de datos podría incluir la dirección de multidifusión de IP. Dado que la dirección de

multidifusión de IP es parte de la cabecera de cada paquete de multidifusión de IP, las observaciones hechas anteriormente acerca de los paquetes del UDP se aplican asimismo a esta situación.

Se prefiere la formación de claves a partir de la posición del símbolo de salida, cuando es posible. La formación de claves por posición podría funcionar bien para leer símbolos de salida desde un dispositivo de almacenamiento, tal como un CD-ROM (Disco Compacto de Memoria de Sólo Lectura), donde la clave de un símbolo de salida es su posición en el CD-ROM (es decir, pista, más sector, más ubicación dentro del sector, etc.). La formación de claves por posición también podría funcionar bien para un sistema de transmisión basado en circuitos, tal como un sistema de ATM (Modalidad de Transferencia Asíncrona), donde las células ordenadas de los datos se transmiten bajo estrechas restricciones de temporización. Con esta forma de formación de claves, el destinatario puede recrear la clave de un símbolo de salida, sin requerirse ningún espacio para transmitir explícitamente la clave. La formación de claves por posición, por supuesto, requiere que tal información de posición esté disponible y sea fiable.

La formación de claves por posición también podría combinarse con otros procedimientos de formación de claves. Por ejemplo, consideremos un sistema de transmisión de paquetes donde cada paquete contiene más de un símbolo de salida. En este caso, la clave del símbolo de salida podría ser construida a partir de un único identificador de flujo, un número de secuencia y la posición del símbolo de salida dentro del paquete. Dado que los borrados de datos generalmente dan como resultado la pérdida de paquetes enteros, el destinatario recibe generalmente un paquete completo. En este caso, el destinatario puede recrear la clave de un símbolo de salida a partir de la cabecera del paquete (que contiene un único identificador de flujo), el número de secuencia en el paquete y la posición del símbolo de salida dentro del paquete.

Otra forma de formación de claves que es preferida en algunos sistemas es la formación de claves al azar. En estos sistemas, se genera un número aleatorio (o seudo-aleatorio), usado como la clave para cada símbolo de salida, y explícitamente transmitido con el símbolo de salida. Una propiedad de la formación de claves al azar es que es probable que la fracción de las claves que tienen el mismo valor sea pequeña, incluso para claves generadas por distintos remitentes en distintas ubicaciones físicas (suponiendo que la gama de posibles claves sea bastante grande). Esta forma de formación de claves puede tener ventaja sobre otras formas en algunos sistemas, debido a la simplicidad de su implementación.

Como se ha explicado anteriormente, la codificación por reacción en cadena es útil allí donde se espera un borrado de datos o donde el destinatario no comienza y termina la recepción exactamente cuando una transmisión comienza y termina. La última condición se denomina en el presente documento “incompletitud de datos”. Estas condiciones no afectan adversamente al proceso de comunicación cuando se usa la codificación por reacción en cadena, porque los datos de codificación por reacción en cadena que son recibidos son sumamente independientes, de modo que sean aditivos en información. Si la mayoría de las colecciones aleatorias de símbolos de salida son bastante independientes como para ser aditivas en información en gran medida, que es el caso para los sistemas de codificación por reacción en cadena descritos en el presente documento, entonces cualquier número adecuado de paquetes puede ser usado para recuperar un fichero de entrada. Si se pierden un centenar de paquetes debido a una ráfaga de ruido que causa el borrado de datos, puede recogerse un centenar extra de paquetes después de la ráfaga, para sustituir la pérdida de los paquetes borrados. Si se pierden miles de paquetes porque un receptor no se sintonizó con un transmisor cuando éste empezó a transmitir, el receptor podría simplemente recoger esos miles de paquetes desde cualquier otro periodo de transmisión, o incluso desde otro transmisor. Con la codificación por reacción en cadena, un receptor no está constreñido a recoger cualquier tipo específico de paquetes, por lo que puede recibir algunos paquetes desde un transmisor, conmutar a otro transmisor, perder algunos paquetes, perderse el principio o el fin de una transmisión dada y sin embargo recuperar un fichero de entrada. La capacidad de incorporarse a, y abandonar, una transmisión sin coordinación entre receptor y transmisor simplifica en gran medida el proceso de comunicación.

Una implementación básica

La transmisión de un fichero usando la codificación por reacción en cadena implica generar, formar o extraer símbolos de entrada a partir de un fichero de entrada, codificar esos símbolos de entrada en uno o más símbolos de salida, donde cada símbolo de salida es generado en base a su clave, independientemente de todos los otros símbolos de salida, y transmitir los símbolos de salida a uno o más destinatarios por un canal. La recepción (y reconstrucción) de una copia del fichero de entrada usando la codificación por reacción en cadena implica recibir algún conjunto, o subconjunto, de símbolos de salida desde uno o más flujos de datos, y descodificar los símbolos de entrada a partir de los valores y claves de los símbolos de salida recibidos.

Como se explicará, el descodificador puede recuperar un símbolo de entrada a partir de los valores de uno o más símbolos de salida y, posiblemente, a partir de información acerca de los valores de otros símbolos de entrada que ya han sido recuperados. De esta manera, el descodificador puede recuperar algunos símbolos de entrada a partir de algunos símbolos de salida, lo que a su vez permite al descodificador descodificar otros símbolos de entrada a partir de esos símbolos de entrada descodificados y símbolos de salida previamente recibidos, y así sucesivamente, causando de este

modo una “reacción en cadena” de recuperación de símbolos de entrada de un fichero que está siendo reconstruido en el destinatario.

Se describirán ahora aspectos de la invención con referencia a las figuras.

La Fig. 1 es un diagrama en bloques de un sistema 100 de comunicaciones que usa la codificación por reacción en cadena. En el sistema 100 de comunicaciones, un fichero 101 de entrada, o un flujo 105 de entrada, es proporcionado a un generador 110 de símbolos de entrada. El generador 110 de símbolos de entrada genera una secuencia de uno o más símbolos de entrada (IS(0), IS(1), IS(2), ...) a partir del fichero o flujo de entrada, teniendo cada símbolo de entrada un valor y una posición (indicado en la Fig. 1 como un entero entre paréntesis). Como se ha explicado anteriormente, los posibles valores para los símbolos de entrada, es decir, su alfabeto, son habitualmente un alfabeto de 2^M símbolos, de modo que cada símbolo de entrada codifica M bits del fichero de entrada. El valor de M está generalmente determinado por el uso del sistema 100 de comunicación, pero un sistema de propósito general podría incluir una entrada del tamaño del símbolo para el generador 110 de símbolos de entrada, de modo que M pueda variar entre uso y uso. La salida del generador 110 de símbolos de entrada se proporciona a un codificador 115.

El generador 120 de claves genera una clave para cada símbolo de salida a ser generado por el codificador 115. Cada clave es generada según uno de los procedimientos descritos anteriormente, o cualquier procedimiento comparable que garantice que una gran fracción de las claves generadas para el mismo fichero de entrada sean únicas, ya sea que estén generadas por este u otro generador de claves. Por ejemplo, el generador 120 de claves puede usar una combinación de la salida de un contador 125, un identificador 130 único de flujo y / o la salida de un generador aleatorio 135 para producir cada clave. La salida del generador 120 de claves se proporciona al codificador 115.

A partir de cada clave I proporcionada por el generador 120 de claves, el codificador 115 genera un símbolo de salida, con un valor B(I), a partir de los símbolos de entrada proporcionados por el generador de símbolos de entrada. El valor de cada símbolo de salida es generado en base a su clave y a alguna función de uno o más de los símbolos de entrada, mencionados en el presente documento como los “símbolos de entrada asociados” del símbolo de salida o simplemente sus “asociados”. La selección de la función (la “función de valor”) y de los asociados se hace según un proceso descrito en más detalle más adelante. Habitualmente, pero no siempre, M es el mismo para símbolos de entrada y símbolos de salida, es decir, ambos se codifican para el mismo número de bits.

En algunas realizaciones, el número K de símbolos de entrada es usado por el codificador para seleccionar los asociados. Si K no es conocido de antemano, tal como allí donde la entrada es un fichero de transmisión por flujo, K puede ser solamente una estimación. El valor K también podría ser usado por el codificador 115 para adjudicar almacenamiento para símbolos de entrada.

El codificador 115 proporciona símbolos de salida a un módulo 140 de transmisión. Al módulo 140 de transmisión también se proporciona la clave de cada símbolo de salida de ese tipo desde el generador 120 de claves. El módulo 140 de transmisión transmite los símbolos de salida y, según el procedimiento de formación de claves usado, el módulo 140 de transmisión podría también transmitir algunos datos acerca de las claves de los símbolos de salida transmitidos, por un canal 145, a un módulo receptor 150. Se supone que el canal 145 es un canal de borrado, pero ese no un requisito para el funcionamiento adecuado del sistema 100 de comunicación. Los módulos 140, 145 y 150 pueden ser componentes adecuados de hardware cualesquiera, componentes de software, medios físicos o cualquier combinación de los mismos, mientras el módulo 140 de transmisión esté adaptado para transmitir símbolos de salida y cualquier dato necesario acerca de sus claves al canal 145, y el módulo 150 de recepción esté adaptado para recibir símbolos y, potencialmente, algunos datos acerca de sus claves desde el canal 145. El valor de K, si se usa para determinar los asociados, puede ser enviado por el canal 145, o bien puede ser fijado por adelantado, por acuerdo del codificador 115 y el descodificador 155.

Como se ha explicado anteriormente, el canal 145 puede ser un canal en tiempo real, tal como un trayecto a través de Internet o un enlace de difusión desde un transmisor de televisión a un destinatario de televisión, o una conexión telefónica desde un punto a otro, o bien el canal 145 puede ser un canal de almacenamiento, tal como un CD-ROM, controlador de disco, sede de Internet o similares. El canal 145 podría ser incluso una combinación de un canal en tiempo real y un canal de almacenamiento, tal como un canal formado cuando una persona transmite un fichero de entrada desde un ordenador personal a un Proveedor de Servicios de Internet (ISP) por una línea telefónica, el fichero de entrada es almacenado en un servidor de la Red y es posteriormente transmitido a un destinatario por Internet.

Debido a que se supone que el canal 145 es un canal de borrado, el sistema 100 de comunicaciones no supone una correspondencia de uno a uno entre los símbolos de salida que salen del módulo receptor 150 y los símbolos de salida que van al módulo transmisor 140. De hecho, allí donde el canal 145 comprende una red de paquetes, el sistema 100 de comunicaciones podría incluso no ser capaz de suponer que el orden relativo de dos o más paquetes cualesquiera se preserve en tránsito a través del canal 145. Por lo tanto, la clave de los símbolos de salida se determina usando uno o más de los esquemas de formación de claves descritos anteriormente, y no necesariamente está determinada por el

orden en el cual los símbolos de salida salen del módulo receptor 150.

El módulo receptor 150 proporciona los símbolos de salida a un descodificador 155, y cualquier dato que el módulo receptor 150 recibe acerca de las claves de estos símbolos de salida es proporcionado a un regenerador 160 de claves. El regenerador 160 de claves regenera las claves para los símbolos de salida recibidos y proporciona estas claves al descodificador 155. El descodificador 155 usa las claves proporcionadas por el regenerador 160 de claves, junto con los correspondientes símbolos de salida, para recuperar los símbolos de entrada (nuevamente IS(0), IS(1), IS(2), ...). El descodificador 155 proporciona los símbolos de entrada recuperados a un reensamblador 165 de ficheros de entrada, que generar una copia 170 del fichero 101 de entrada o del flujo 105 de entrada.

Un codificador básico

La Fig. 2 es un diagrama en bloques de una realización del codificador 115 mostrado en la Fig. 1. El diagrama en bloques de la Fig. 2 se explica en el presente documento con referencias a la Fig. 3, que es un diagrama que muestra el equivalente lógico de algo del procesamiento realizado por el codificador mostrado en la Fig. 2.

Al codificador 115 se proporcionan los símbolos de entrada y una clave para cada símbolo de salida que ha de generar. Según se muestra, los K símbolos de entrada son almacenados en un almacén temporal 205 de símbolos de entrada. La clave I (proporcionada por el generador 120 de claves mostrado en la Fig. 1) es una entrada para el selector 210 de funciones de valor, el selector 215 de pesos y el asociador 220. El número K de símbolos de entrada también es proporcionado a estos tres componentes 210, 215 y 220. Un calculador 225 está acoplado para recibir salidas desde el selector 210 de funciones de valor, el selector 215 de pesos, el asociador 220 y el almacén temporal 205 de símbolos de entrada, y tiene una salida para los valores de símbolos de salida. Debería entenderse que podrían usarse otras disposiciones equivalentes para los elementos mostrados en la Fig. 2, y que esto no es más que un ejemplo de un codificador según la presente invención.

En funcionamiento, los K símbolos de entrada son recibidos y almacenados en el almacén temporal 205 de símbolos de entrada. Como se ha explicado anteriormente, cada símbolo de entrada tiene una posición (es decir, su posición original en el fichero de entrada) y un valor. Los símbolos de entrada no necesariamente deben ser almacenados en el almacén temporal 205 de símbolos de entrada en su respectivo orden, mientras pueda determinarse la posición de los símbolos de entrada almacenados.

Usando la clave I y el número K de símbolos de entrada, el selector 215 de pesos determina el número W(I) de símbolos de entrada que han de ser “asociados” del símbolo de salida con clave I. Usando la clave I, el peso W(I) y el número K de símbolos de entrada, el asociador 220 determina la lista AL(I) de posiciones de símbolos de entrada asociados al símbolo de salida. Debería entenderse que W(I) no necesariamente ha de ser calculado por separado, o explícitamente, si el asociador 220 puede generar AL(I) sin conocer W(I) de antemano. Una vez que AL(I) está generado, W(I) puede ser fácilmente determinado, porque es el número de asociados en AL(I).

Una vez que I, W(I) y AL(I) son conocidos, el valor B(I) del símbolo de salida es calculado por el calculador 225, en base a una función F(I) de valor. Una propiedad de una función de valor adecuada es que permita que el valor para un asociado en AL(I) sea determinado a partir del valor B(I) de símbolo de salida y a partir de los valores para los otros W(I)-1 asociados en AL(I). Una función de valor preferida usada en esta etapa es la función de valor XOR, dado que satisface esta propiedad, se calcula fácilmente y se invierte fácilmente. Sin embargo, podrían ser usadas en cambio otras funciones de valor adecuadas.

Si se usa, el selector 210 de función de valor determina una función F(I) de valor a partir de la clave I y a partir de K. En una variación, la función F(I) de valor es la misma función F de valor para todo I. En esa variación, el selector 210 de función de valor no es necesario y el calculador 225 puede ser configurado con la función F de valor. Por ejemplo, la función de valor podría ser XOR para todo I, es decir, el valor del símbolo de salida es un XOR (O exclusivo) de los valores de todos sus asociados.

Para cada clave I, el selector 215 de pesos determina un peso W(I) a partir de I y K. En una variación, el selector 215 de pesos selecciona W(I) usando la clave I para generar primero un número de aspecto aleatorio, y luego usa este número para buscar el valor de W(I) en una tabla de distribución que está almacenada dentro del selector 215 de pesos. Una descripción más detallada de cómo una tabla de distribución de ese tipo podría formarse y accederse a ella se da más adelante. Una vez que el selector 215 de pesos determina W(I), este valor se proporciona al asociador 220 y al calculador 225.

El asociador 220 determina una lista AL(I) de las posiciones de los W(I) símbolos de entrada asociados al símbolo de salida actual. La asociación se basa en el valor de I, en el valor de W(I) y en K (si está disponible). Una vez que el asociador 220 determina AL(I), el AL(I) se proporciona al calculador 225. Usando la lista AL(I), el peso W(I) y la función F(I) de valor proporcionada por el selector 210 de función de valor, o bien una función F de valor preseleccionada, el calculador 225 accede a los W(I) símbolos de entrada mencionados por AL(I) en el almacén temporal 205 de símbolos de

entrada para calcular el valor B(I) para el símbolo de salida actual. Un ejemplo de un procedimiento para calcular A(I) se da más adelante, pero podría usarse en cambio otro procedimiento adecuado. Preferiblemente, el procedimiento da a cada símbolo de entrada una oportunidad aproximadamente igual de ser seleccionado como un asociado para un símbolo de salida dado, y realiza la selección de una manera que el descodificador pueda reproducir si el descodificador no tiene ya AL(I) disponible para el mismo.

El codificador 115 emite luego B(I). En efecto, el codificador 115 realiza la acción ilustrada en la Fig. 3, esto es, generar un valor B(I) de símbolo de salida como alguna función de valor de símbolos de entrada seleccionados. En el ejemplo mostrado, la función de valor es XOR, el peso W(I) del símbolo de salida es 3 y los símbolos de entrada asociados (los asociados) están en las posiciones 0, 2 y 3 y tienen los respectivos valores IS(0), IS(2) e IS(3). Así, el símbolo de salida se calcula como:

B(I) = IS(0) XOR IS(2) XOR IS(3)

para ese valor de I.

Los símbolos de salida generados se transmiten luego y son recibidos según lo descrito anteriormente. En el presente documento, se supone que algunos de los símbolos de salida podrían haberse perdido o desordenado, o bien fueron generados por uno o más codificadores. Se supone, sin embargo, que los símbolos de salida que son recibidos han sido recibidos con una indicación de su clave y alguna garantía de que sus valores de B(I) son exactos. Según se muestra en la Fig. 1, esos símbolos de salida recibidos, junto con sus correspondientes claves reconstruidas a partir de su indicación por el regenerador 160 de claves y el valor de K, son la entrada al descodificador 155.

El número de bits, M, codificado en un símbolo de entrada (es decir, su tamaño) depende de la aplicación. El tamaño de un símbolo de salida también depende de la aplicación, pero también podría depender del canal. Por ejemplo, si el típico fichero de entrada es un fichero de múltiples megaoctetos, el fichero de entrada podría ser partido en miles, decenas de miles o cientos de miles de símbolos de entrada, codificando cada símbolo de entrada unas pocas decenas, centenas o miles de octetos.

En algunos casos, el proceso de codificación podría ser simplificado si los valores de símbolos de salida y los valores de símbolos de entrada tuvieran el mismo tamaño (es decir, fueran representables por el mismo número de bits o seleccionados a partir del mismo alfabeto). Si es ese el caso, entonces el tamaño del valor del símbolo de entrada está limitado cuando el tamaño del valor del símbolo de salida está limitado, tal como cuando se desea poner símbolos de salida en paquetes y cada símbolo de salida debe caber en un paquete de tamaño limitado. Si algunos datos acerca de la clave hubieran de ser transmitidos a rin de recuperar la clave en el receptor, el símbolo de salida sería preferiblemente lo bastante pequeño como para admitir el valor y los datos acerca de la clave en un paquete.

Como se ha descrito anteriormente, aunque las posiciones de los símbolos de entrada son habitualmente consecutivas, en muchas implementaciones las claves están lejos de ser consecutivas. Por ejemplo, si un fichero de entrada se dividiera en hasta 60.000 símbolos de entrada, las posiciones para los símbolos de entrada variarían entre 0 y 59.999, mientras que en una de las implementaciones mencionadas anteriormente cada clave podría ser escogida independientemente como un número aleatorio de 32 bits y los símbolos de salida podrían ser generados continuamente y transmitidos hasta que el remitente se detenga. Según se muestra en el presente documento, la codificación por reacción en cadena permite que el fichero de entrada de 60.000 símbolos sea reconstruido a partir de cualquier colección suficientemente grande (60.000 + algún incremento A) de símbolos de salida, independientemente de dónde fueron tomados esos símbolos de salida en la secuencia de salida.

Un descodificador básico

La Fig. 4 muestra una realización del descodificador 155 en detalle, con muchas partes en común con el codificador 115 mostrado en la Fig. 5. El descodificador 155 comprende un selector 210 de función de valor, un selector 215 de peso, un asociador 220, un almacén temporal 405 de símbolos de salida, un reductor 415, un reconstructor 420 y un almacén temporal 425 de reconstrucción. Como ocurre con el codificador, el selector 210 de función de valor y el espacio en el almacén temporal 405 de símbolos de salida, adjudicado para almacenar la descripción de la función de valor, son optativos y podrían no ser usados si la función de valor fuera la misma para todos los símbolos de salida. Se muestran varias entradas del almacén temporal 425 de reconstrucción, con algunos símbolos de entrada reconstruidos y con otros desconocidos de momento. Por ejemplo, en la Fig. 4, los símbolos de entrada en las posiciones 0, 2, 5 y 6 han sido recuperados y los símbolos de entrada en las posiciones 1, 3 y 4 aún han de ser recuperados.

En funcionamiento, para cada símbolo de salida recibido con clave I y valor B(I), el descodificador 155 hace lo siguiente. La clave I es proporcionada al selector 210 de función de valor, el selector 215 de peso y el asociador 220. Usando K y la clave I, el selector 215 de peso determina el peso W(I). Usando K, la clave I y W(I), el asociador 220 produce la lista AL(I) de W(I) posiciones de símbolos de entrada asociados al símbolo de salida. Optativamente, usando K e I, el selector 210 de función de valor selecciona la función F(I) de valor. Luego, I, B(I), W(I) y AL(I), y optativamente F(I), son almacenados

en una fila del almacén temporal 405 de símbolos de salida. El selector 210 de función de valor, el selector 215 de peso y el asociador 220 realizan la misma operación para el descodificador 155, según se ha descrito para el codificador 115. En particular, la función F(I) de valor, el peso W(I) y la lista AL(I) producidos por el selector 210 de función de valor, el selector 215 de peso y por el asociador 220 en la Fig. 5, son los mismos para la misma clave I que para las partes correspondientes mostradas en la Fig. 4. Si K varía de un fichero de entrada a otro, puede ser comunicado desde el codificador al descodificador de cualquier manera convencional, tal como incluyéndolo en una cabecera de mensaje.

El reconstructor 420 examina el almacén temporal 405 de símbolos de salida, buscando símbolos de salida almacenados allí que tengan peso uno, es decir, W(I) = 1, y AL(I) enumera la posición de solamente un asociado. Esos símbolos son mencionados en el presente documento como miembros de un “conjunto descodificable”. Para funciones de valor con las propiedades descritas anteriormente, los símbolos de salida de peso uno están en el conjunto descodificable porque un valor de un símbolo de entrada puede ser determinado a partir de ese símbolo de salida. Por supuesto, si se usara una función de valor que permitiera que los símbolos de entrada fueran descodificados con una condición distinta a tener un peso de uno, esa condición sería usada para determinar si un símbolo de salida está o no en el conjunto descodificable. Para mayor claridad, los ejemplos descritos aquí suponen que el conjunto descodificable es el de aquellos símbolos de salida con peso uno, y las extensiones de estos ejemplos a otras condiciones descodificables por la función de valor deberían ser evidentes a partir de esta descripción.

Cuando el reconstructor 420 halla un símbolo de salida que está en el conjunto descodificable, el valor B(I) del símbolo de salida y, optativamente, la función F(I) de valor, es usado para reconstruir el símbolo de entrada enumerado en AL(I), y el símbolo de entrada reconstruido es colocado en el almacén temporal 425 de reconstrucción en la posición adecuada para ese símbolo de entrada. Si el símbolo de entrada indicado ya había sido reconstruido, el reconstructor 420 podría descartar el símbolo de entrada recientemente reconstruido, sobrescribir el símbolo de entrada reconstruido ya existente, o comparar los dos y emitir un error si difieren. Allí donde la función de valor es un XOR de todos los asociados, el valor del símbolo de entrada es sencillamente el valor del símbolo de salida. El reconstructor 420 reconstruye así símbolos de entrada, pero solamente a partir de símbolos de salida en el conjunto descodificable. Una vez que un símbolo de salida proveniente del conjunto descodificable es usado para reconstruir un símbolo de entrada, puede ser borrado para ahorrar espacio en el almacén temporal 405 de símbolos de salida. El borrado del símbolo de salida “consumido” también asegura que el reconstructor 420 no revisite continuamente ese símbolo de salida.

Inicialmente, el reconstructor 420 espera hasta que sea recibido al menos un símbolo de salida que sea un miembro del conjunto descodificable. Una vez que ese símbolo de salida sea usado, el conjunto descodificable quedaría vacío otra vez, excepto por el hecho de que algún otro símbolo de salida podría ser una función de solamente ese símbolo de entrada reconstruido y algún otro símbolo de entrada. Así, reconstruir un símbolo de entrada a partir de un miembro del conjunto descodificable podría causar que otros símbolos de salida fueran añadidos al conjunto descodificable. El proceso de reducción de símbolos de salida para añadirlos al conjunto descodificable es realizado por el reductor 415.

El reductor 415 recorre el almacén temporal 405 de símbolo de salida y el almacén temporal 425 de reconstrucción para hallar símbolos de salida que tengan listas AL(I) que enumeren posiciones de símbolos de entrada que hayan sido recuperados. Cuando el reductor 415 halla un tal símbolo de salida “reducible” con clave I, el reductor 415 obtiene el valor IS(R) de un símbolo de entrada recuperado en la posición R y modifica B(I), W(I) y AL(I) de la siguiente manera:

B(I) se reinicia con B(I) XOR IS(R)

W(I) se reinicia con W(I) - 1

AL(I) se reinicia con AL(I), excluyendo a R

En las ecuaciones anteriores, se supone que la función de valor era un XOR de todos los valores de los asociados. Obsérvese que XOR es su propia inversa – si no fuera ese el caso y se usara originalmente otra función de valor para calcular el símbolo de salida, entonces la inversa de esa función de valor sería usada aquí por el reductor 415. Como debería ser evidente, si se conocen los valores para más de un asociado, el equivalente de las ecuaciones anteriores puede ser calculado para hacer que B(I) dependa solamente de valores asociados desconocidos cualesquiera (y ajustar W(I) y L(I) en consecuencia).

La acción del reductor 415 reduce los pesos de los símbolos de salida en el almacén temporal 405 de símbolos de salida. Cuando el peso de un símbolo de salida es reducido a uno (o bien ocurre otra condición descodificable para otras funciones de valor), entonces ese símbolo de salida se convierte en un miembro del conjunto descodificable, sobre el que puede actuar luego el reconstructor 420. En la práctica, una vez que se ha recibido un número suficiente de símbolos de salida, el reductor 415 y el reconstructor 420 crean una descodificación por reacción en cadena, descodificando el reconstructor 420 el conjunto descodificable para recuperar más símbolos de entrada, usando el reductor 415 esos símbolos de entrada recién recuperados para reducir más símbolos de salida, a fin de que sean añadidos al conjunto descodificable, y así sucesivamente, hasta que todos los símbolos de entrada provenientes del fichero de entrada estén recuperados.

El descodificador mostrado en la Fig. 4 reconstruye símbolos de entrada de manera directa, sin mucha consideración para el almacenamiento en memoria, los ciclos de cálculo o el tiempo de transmisión. Allí donde la memoria del descodificador, el tiempo de descodificación o el tiempo de transmisión (que restringe el número de símbolos de salida que son recibidos) están limitados, el descodificador puede ser optimizado para usar mejor esos recursos limitados.

Un descodificador más eficaz

La Fig. 5 muestra una realización preferida de una implementación más eficaz de un descodificador 500 en detalle. Aquí, se supone que la función de valor es XOR. Valen implementaciones similares con respecto a funciones de valor distintas a XOR. Con referencia a la Fig. 5, el descodificador 500 comprende la estructura 505 de datos de símbolos de salida (mencionada a continuación como la OSDS 505), la estructura 510 de datos de símbolos de entrada (mencionada a continuación como la ISDS 510), la pila 515 del conjunto descodificable (mencionada en adelante como la DSS 515), el organizador 520 de recepción y el procesador 525 de recuperación.

La OSDS 505 es una tabla que almacena información acerca de símbolos de salida, donde la fila R de la OSDS 505 almacena información acerca del R-ésimo símbolo que se ha recibido. Una variable R mantiene el rastro del número de símbolos de salida que han sido recibidos, y se inicializa con cero. La OSDS 505 almacena los campos CLAVE, VALOR, PESO y XOR_POS para cada fila, con los campos mostrados organizados en columnas. El campo CLAVE almacena la clave del símbolo de salida. El campo VALOR almacena el valor del símbolo de salida, que es actualizado durante el procesamiento. Todos los símbolos de salida que son eventualmente usados para recuperar un símbolo de entrada tienen eventualmente su VALOR modificado con el valor del símbolo de entrada recuperado. El campo PESO almacena el peso inicial del símbolo de salida. El PESO de un símbolo de salida se reduce a lo largo del tiempo hasta que se hace uno y luego puede ser usado para recuperar un símbolo de entrada. El campo XOR_POS almacena inicialmente el valor XOR de todas las posiciones de los asociados del símbolo de salida. Cuando el PESO de un símbolo de salida se hace uno, el XOR_POS del símbolo de salida se convierte en la posición del único asociado restante.

La ISDS 510 es una tabla que almacena información acerca de símbolos de entrada, donde la fila P almacena información acerca del símbolo de entrada en la posición P. Para cada fila, la ISDS 510 incluye almacenamiento para un campo FILA_REC, que eventualmente se convierte en el número de fila en la OSDS 505 del símbolo de salida que es usado para recuperar el símbolo de entrada, un campo ÍND_REC, que se inicializa con todos los valores en “no”, e indica si se han recuperado o no símbolos de entrada, y un campo RL. Cuando se recupera un símbolo de entrada, el ÍND_REC del símbolo de entrada se cambia a “sí”. La columna RL se inicializa con todos los valores en “lista vacía”. Según se reciben símbolos de salida que tengan un símbolo de entrada como un asociado, el número de fila en la OSDS 505 del símbolo de salida se añade a la lista de RL para el símbolo de entrada.

La DSS 515 es una pila que almacena información acerca del conjunto descodificable. Una variable S rastrea el tamaño del conjunto descodificable, y se inicializa con cero. En la DSS 515, la columna FILA_SAL almacena números de fila en la OSDS 505 de símbolos de salida, y la columna POS_ENT almacena las posiciones en la ISDS 510 de los símbolos de entrada que pueden ser recuperados.

En una realización, el descodificador 500 funciona de la siguiente manera, y según se muestra en el diagrama de flujo de la Fig. 6, con las correspondientes etapas de la Fig. 6 indicadas entre paréntesis en la descripción del proceso. Primero, la ISDS 510 es inicializada según lo descrito anteriormente, y tanto R como S son inicializados con cero (605). Cuando se recibe un nuevo símbolo de salida (610), es decir, la clave I y el valor B(I) del símbolo de salida, la CLAVE(R) se fija en I y el VALOR(R) se fija en B(I) en la OSDS 505 (615). El organizador 520 de recepción es llamado luego para procesar el símbolo de salida recibido con la clave I almacenada en la fila R de la OSDS 505 (620). Esto incluye añadir información a la OSDS 505 y a la DSS 515, usando adecuadamente la información almacenada en la ISDS 510, según se muestra en el diagrama de flujo de la Fig. 7. Luego, se incremente R en uno (625) para hacer que el próximo símbolo de salida sea almacenado en la siguiente fila de la OSDS 505. El procesador 525 de recuperación es llamado luego para procesar símbolos de salida en el conjunto descodificable y para añadir nuevos símbolos de salida al conjunto descodificable (630). Esto incluye añadir a, y borrar de, el conjunto descodificable almacenado en la DSS 515, usando y modificando partes de la ISDS 510 y la OSDS 505 adecuadamente, según se muestra en el diagrama de flujo de la Fig. 8 (a) y / o 8(b). El descodificador 500 rastrea el número de símbolos de entrada que han sido recuperados, y cuando este número llega a K, es decir, todos los símbolos de entrada han sido recuperados, el descodificador 500 termina con éxito; en caso contrario, vuelve a la etapa 610 para recibir el próximo símbolo de salida, según se muestra en 635 y 640.

Un diagrama de flujo que describe el funcionamiento del organizador 520 de recepción se muestra en la Fig. 7, que se refiere a las Figs. 9 a 12. Cuando llega un símbolo de salida con valor B(I) y clave I, el organizador 520 de recepción realiza las siguientes operaciones, con referencia a la Fig. 7. El peso W(I) se calcula a partir de I y K (705) y la lista AL(I) de las posiciones de asociados se calcula a partir de I, W(I) y K (710). Las Figs. 11 a 12 muestran los detalles de un cálculo de W(I) y las Figs. 9 a 10 muestran los detalles de un cálculo de AL(I).

Con referencia nuevamente a la Fig. 7, el valor de XL(I) es calculado como el XOR de todas las posiciones en AL(I) (715).

Luego, para cada posición P en la lista AL(I), si el símbolo P de entrada no está recuperado, es decir, si ÍND_REC(P) = "no" en la ISDS 510, entonces R se añade a la lista RL(P) en la ISDS 510; en caso contrario, si el símbolo P de entrada está recuperado, es decir, si ÍND_REC(P) = "sí" en la ISDS 510, entonces W(I) se decrementa en uno y XL(I) se reinicia con XL(I) XOR P (720). Luego, XOR_POS(R) se fija en XL(I) y PESO(R) se fija en W(I) en la OSDS 505 (725). El PESO(R) se compara luego con uno (730). Si PESO(R) es igual a uno, entonces el símbolo de salida es añadido al conjunto descodificable, es decir, FILA_SAL(S) se fija en R y POS_ENT(S) se fija en XOR_POS(R) en la DSS 515, y el valor de S se incrementa en uno (735). Finalmente, el organizador 520 de recepción devuelve el control (740).

Un diagrama de flujo que describe una operación del procesador 525 de recuperación se muestra en la Fig. 8(a), que se refiere a las Figs. 9 a 12. En esa operación, el procesador 525 de recuperación primero comprueba si el conjunto descodificable está vacío, es decir, si S = 0 y, si es así, devuelve el control inmediatamente (805, 810). Si el conjunto descodificable no está vacío, entonces S es decrementado en uno (815) y el número R' de fila del símbolo de salida y la posición P del símbolo de entrada asociado son cargadas desde la DSS 515 (820). Si el símbolo de entrada en la posición P ya ha sido recuperado, es decir, si ÍND_REC(P) = "sí" (825), entonces el procesador 525 de recuperación deja de procesar este elemento del conjunto descodificable y continúa para procesar el próximo. En caso contrario, el símbolo de salida almacenado en el número R' de fila en la OSDS 505 es usado para recuperar el símbolo de entrada en la posición P en la ISDS 510, y esto es indicado fijando ÍND_ REC(P) en "sí" y FILA_REC(P) en R' en la ISDS 510 (830). Luego, la clave original del símbolo de salida, CLAVE(R'), proveniente de la OSDS 505, se carga en I (835) a fin de calcular el peso original W(I) y la lista original de asociados AL(I) de la clave (840, 845).

Con referencia todavía a la Fig. 8(a), el valor recuperado del símbolo de entrada en la posición P es calculado como el XOR del símbolo de salida y todos los asociados del símbolo de salida, excluyendo al símbolo de entrada. Esto se calcula considerando todas las posiciones P' en AL(I) distintas de P. Obsérvese que FILA_REC(P') en la ISDS 510 almacena el número de fila del valor recuperado para el símbolo de entrada en la posición P', y que VALOR(FILA_REC(P')) en la OSDS 505 es este valor recuperado. Este cálculo se muestra en 850 de la Fig. 8(a), y el valor recuperado para el símbolo de entrada en la posición P está almacenado en VALOR(R') = VALOR(FILA_REC(P)) al final de este cálculo.

Una variación del proceso mostrado en la Fig. 8(a) se muestra en la Fig. 8(b). Allí, en lugar de llevar a cabo las etapas 830, 835, 840, 845 y 850 para cada símbolo de salida según es procesado, los valores de R' y P pueden ser almacenados en una planificación de ejecución para un procesamiento posterior (865). Un ejemplo de procesamiento de ejecución diferida se muestra en la Fig. 8(c), incluyendo etapas mencionadas como 870, 875, 880 y 885. En esta variación, el diagrama de flujo mostrado en la Fig. 6 se modifica inicializando E con cero en la etapa 605. El procesamiento diferido de la planificación de ejecución puede ocurrir después de que el descodificador determine que los símbolos recibidos son suficientes para descodificar el fichero entero, p. ej., en la etapa 640, después de que se sabe que todos los símbolos de entrada son recuperables. En algunos casos, especialmente allí donde los símbolos de entrada son grandes, la ejecución de la planificación podría ser diferida hasta que el fichero de entrada, o partes del mismo, se necesiten en el receptor.

En cualquier variación, es decir, ya sea en la Fig. 8(a) o en la Fig. 8(b), en la etapa 855 los símbolos de salida que todavía tienen el símbolo P de entrada como asociado son modificados para reflejar que el símbolo de entrada ha sido recuperado. Los números de fila de estos símbolos de salida en la OSDS 505 son almacenados en RL(P). Para cada número R'' de fila en RL(P), el PESO(R'') es decrementado en uno y P es objeto de una operación XOR con resultado en XOR_POS(R''), para reflejar la eliminación del símbolo de entrada en la posición P como asociado del símbolo de salida en la fila R'' de la OSDS 505. Si esta modificación causa que el símbolo de salida en la fila R'' de la OSDS 505 adquiera peso uno, es decir, PESO(R'') = 1, entonces este símbolo de salida es añadido al conjunto descodificable fijando FILA_SAL(S) en R'', POS_ENT(S) en XOR_POS(R'') e incrementando S en uno. Finalmente, el espacio usado para almacenar números de filas de símbolos de salida en la lista RL(P) es devuelto al espacio libre (860), y el procesamiento continúa en la etapa 805.

Una implementación del asociador

La correlación entre una clave de símbolo de salida y símbolos de entrada asociados (es decir, la determinación del peso W(I) y la lista AL(I) de posiciones de asociados para una clave I) puede tomar una gran variedad de formas. W(I) debería ser escogido con el mismo valor tanto por el codificador como por el descodificador, para la misma clave I (en el remitente y el destinatario, respectivamente). De manera similar, AL(I) debería ser escogido para que contenga la misma lista de posiciones, tanto por el codificador como por el descodificador, para la misma clave I. El asociador es el objeto que calcula o genera AL(I) a partir de I y, usualmente, W(I) y K.

En una realización, W(I) y AL(I) están determinados de una manera diseñada para imitar un proceso aleatorio. Para satisfacer el requisito de que el codificador y el descodificador produzcan el mismo resultado con respecto a la misma clave, una secuencia seudo-aleatoria podría ser generada tanto por el codificador como por el descodificador, alimentados con la clave. En lugar de una secuencia seudo-aleatoria, podría usarse una secuencia verdaderamente aleatoria para la generación de W(I) y / o AL(I) pero, para que eso sea útil, la secuencia aleatoria usada para generar W(I) y AL(I) debería ser comunicada al destinatario.

En el descodificador mostrado en la Fig. 4, el almacén temporal 405 de símbolos de salida requiere almacenamiento para cada lista de posiciones de asociados del símbolo de salida, es decir, almacenamiento en la columna etiquetada como AL(I). El descodificador más eficaz mostrado en la Fig. 5 no requiere este almacenamiento, porque una lista de asociados es recalculada según se necesita, p. ej., según se muestra en las Figs. 9 a 10. Hay una ventaja en recalcular listas de asociados cada vez, a fin de ahorrar almacenamiento, solamente si estos cálculos pueden ser hechos rápidamente según sea necesario.

Una realización preferida del asociador 220 se muestra en la Fig. 9 y funciona según el proceso mostrado en la Fig. 10. Este asociador puede ser usado en el codificador, así como en el descodificador. Aunque el almacenamiento de memoria para AL(I) en el codificador no es de gran importancia, porque el codificador no necesita normalmente almacenar más de un AL(I) a la vez, el mismo proceso debería ser usado tanto en el codificador como en el descodificador, para asegurar que los valores para AL(I) sean los mismos en ambos lugares.

La entrada al asociador 220 es una clave I, el número K de símbolos de entrada y un peso W(I). La salida es una lista AL(I) de las W(I) posiciones de asociados del símbolo de salida con clave L. Como se muestra en la Fig. 9, el asociador comprende una tabla RBITS_ASOC 905 de bits aleatorios y un calculador CALC_ASOC 910. Antes de que sea generado un AL(I) específico, el tamaño del fichero de entrada es ajustado de modo que el número de símbolos de entrada sea primo. Así, si el fichero de entrada comienza por K símbolos de entrada, el número primo más pequeño, P, mayor o igual que K, es identificado. Si P es mayor que K, se añaden P-K símbolos de entrada en blanco (p. ej., fijados en cero) al fichero de entrada y K se reinicia con P. Para este fichero de entrada modificado, las listas AL(I) de las posiciones de asociados son calculadas según se muestra en las Fig. 9 y 10.

En esta realización, CALC_ASOC 910 funciona según lo descrito más adelante y según se muestra en el diagrama de flujo de la Fig. 10. La primera etapa es usar la clave I, el número K de símbolos de entrada y la tabla de bits aleatorios RBITS_ASOC 905 para generar dos valores enteros X e Y que tenga la propiedad de que X es al menos uno y a lo sumo K-1, e Y es al menos cero y a lo sumo K-1 (1005). Preferiblemente, X e Y están independiente y uniformemente distribuidos dentro de sus respectivas gamas. Luego, un vector V[] con W(I) entradas es inicializado para el almacenamiento de AL(I) según se calculan sus miembros (1010). Dado que V[] es solamente almacenamiento temporal para una lista, ocuparía mucha menos memoria que la columna AL(I) del almacén temporal 405 de símbolos de salida (véase la Fig. 4). V[0] (este es el primer elemento de la lista AL(I)) se fija en Y (1015). Luego, para todos los valores de J a partir de 1 y hasta llegar a W(I)-1, el valor de V[J] se fija en (V[J-1] + X) mod K, según se muestra en las etapas 1020 a 1050. Dado que K es primo y W(I) es a lo sumo K, todos los valores V[] serán únicos. Según se muestra, la operación "mod K" puede ser una sencilla operación de comparación y resta, es decir, las etapas 1035 y 1040. Así, el proceso de producir la lista de posiciones de asociados de un símbolo de salida dado es muy eficaz.

Una ventaja del enfoque anterior para calcular AL(I) es que produce suficiente variedad en la distribución de las posiciones de los asociados, para garantizar que el algoritmo de descodificación funciona con alta probabilidad y con mínimo sobregasto de recepción (es decir, el fichero de entrada es recuperado después de recibir solamente poco más de K símbolos de salida, suponiendo que los símbolos de entrada y los símbolos de salida tengan la misma longitud) cuando se acopla con un buen procedimiento para seleccionar W(I).

Una implementación del selector de pesos

Las prestaciones y la eficacia del codificador / descodificador dependen de la distribución de pesos, y algunas distribuciones son mejores que otras. Los aspectos operativos de la selección de pesos se exponen más adelante, seguidos por una descripción de algunas distribuciones de pesos importantes. El diagrama en bloques de la Fig. 11 y el diagrama de flujo de la Fig. 1 se usan para ilustrar estos conceptos.

Según se muestra en la Fig. 11, el selector de pesos comprende dos procesos INIC_WT1105 y CALC_WT 1110, y dos tablas RBITS_WT 1115 y DISTRIB_WT 1120. El proceso INIC_WT 1105 es invocado solamente una vez cuando la primera clave es introducida para inicializar la tabla DISTRIB_WT 1120. El diseño de DISTRIB_WT 1120 es un aspecto importante del sistema, y se considera más adelante en mucho más detalle. El proceso CALC_WT 1110 es invocado en cada llamada para producir un peso W(I) en base a una clave I. Según se muestra en el diagrama de flujo de la Fig. 12, CALC_WT 1110 usa la clave y los bits aleatorios almacenados en la tabla RBITS_WT para generar un número aleatorio R (1205). Luego el valor de R es usado para seleccionar un número L de fila en la tabla DISTRIB_WT 1120.

Según se muestra en la Fig. 11, las entradas en la columna GAMA de DISTRIB_WT 1120 son una secuencia creciente de enteros positivos que terminan en el valor VAL_MAX. El conjunto de valores posibles para R son los enteros entre cero y VAL_MAX-1. Una propiedad deseable es que R sea, con igual probabilidad, cualquier valor en la gama de valores posibles. El valor de L se determina buscando en la columna GAMA hasta que se halle un L que satisfaga GAMA(L-1) $\leq$ R < GAMA(L) (1210). Una vez que se halla un L, el valor de W(I) se fija en WT(L), y este es el peso devuelto (1215, 1220). En la Fig. 11, para la tabla ejemplar mostrada, si R es igual a 38.500, entonces se halla que L es 4 y, así, W(I) se fija en WT(4) = 8.

Otras variaciones de la implementación de un selector de pesos y un asociador incluyen generar I de manera seudo-aleatoria y generar W(I) y AL(I) directamente a partir de I. Como debería ser evidente, W(I) puede ser determinado examinando AL(I), dado que W(I) es igual al número de asociados en AL(I). Debería ser evidente a partir de esta descripción que muchos otros procedimientos de generar valores de W(I) son equivalentes al sistema recién descrito para generar un conjunto de valores de W(I) con la distribución definida por DISTRIB_WT.

Un descodificador alternativo

Tras leer esta revelación, debería quedar claro a los expertos en la técnica que un receptor puede funcionar más eficazmente que las implementaciones descritas anteriormente. Por ejemplo, el receptor podría ser más eficaz si almacenara temporalmente paquetes y aplicara solamente la regla de recuperación una vez que llegara un grupo de paquetes. Esta modificación reduce el tiempo de cálculo gastado en hacer operaciones posteriormente innecesarias y reduce el sobregasto debido a la conmutación de contexto. En particular, dado que el descodificador no puede esperar recuperar el fichero original de K símbolos de entrada antes de que al menos K símbolos de salida (suponemos símbolos de entrada y de salida del mismo tamaño) lleguen en K paquetes (suponemos un símbolo por paquete), es ventajoso esperar hasta que al menos K paquetes lleguen antes de comenzar el proceso de descodificación.

La Fig. 13 muestra un procedimiento distinto de descodificación, que incluye los conceptos expresados anteriormente y que es una modificación del proceso usado por el descodificador de la Fig. 6. La diferencia primaria entre los dos procedimientos es que el procedimiento de la Fig. 13 recibe símbolos de salida en lotes, según se muestra en 1315. El tamaño del primer lote se fija en K + A, donde A es una pequeña fracción del número K de símbolos de entrada (1310). Después de que se recibe el primer lote de símbolos de salida, los símbolos de salida son procesados igual que antes, usando el organizador 520 de recepción (1340) para procesar los símbolos de salida entremezclados con el uso del procesador 525 de recuperación (1350) para procesar el conjunto descodificable y recuperar símbolos de entrada a partir de símbolos de salida de peso uno reducido. Si la recuperación de todos los K símbolos de entrada no se logra usando el primer lote de K + A símbolos de salida, entonces lotes adicionales de G símbolos de salida son recibidos y procesados hasta que todos los símbolos de entrada sean recuperados.

Es ventajoso minimizar el almacenamiento requerido para las estructuras de datos auxiliares del descodificador, tanto como sea posible. Como ya se ha descrito, el almacenamiento para la lista de asociados para cada símbolo de salida no es necesario, dado que el asociador 220 puede ser usado para calcular rápidamente esas listas, según se necesite. Otra necesidad de almacenamiento es para almacenar, para cada símbolo de entrada no recuperado de momento, el número de fila en la OSDS 505 de los símbolos de salida que tengan el símbolo de entrada como un asociado, es decir, el espacio para las listas mostradas en la columna RL en la tabla ISDS 510 de la Fig. 5. Como ya se ha descrito en la etapa 855 de la Fig. 8, un uso de este almacenamiento es poder identificar rápidamente cuáles símbolos de salida son reducibles cuando se reconstruye un símbolo de entrada dado. A menos que se haga eficazmente, el almacenamiento requerido para estas listas sería proporcional al número total de asociados de todos los símbolos de salida usados para recuperar todos los símbolos de entrada.

Un descodificador de preclasificación

Una realización más preferida del descodificador se describe ahora, con referencia a la Fig. 14 y a la Fig. 15. La Fig. 14 muestra las partes que comprenden el descodificador, que son las mismas que las mostradas en la Fig. 5, excepto por la adición de una tabla CLASIFICACIÓN DE PESOS 1405 y LISTA DE EJECUCIÓN 1420, usadas para almacenar la planificación de ejecución, formada según lo descrito en la Fig. 8(b). La tabla CLASIFICACIÓN DE PESOS se usa para almacenar lotes de números de filas en la OSDS 505 de símbolos de salida, según son recibidos, clasificados en orden de peso creciente. La columna VAL_WT se usa para almacenar pesos, y la columna LISTA_FILAS se usa para almacenar números de filas de símbolos de salida en la OSDS 505. En general, los números de filas de todos los símbolos de salida con peso VAL_WT(L) son almacenados en LISTA_FILAS(L). Esta tabla se usa para procesar el lote de símbolos de salida en orden de peso creciente, según se muestra en la etapa 1520 de la Fig. 15. Los símbolos de salida de peso bajo son menos intensivos en términos de cálculo, para su uso a fin de recuperar un símbolo de entrada, y es probable, según llegan los símbolos de salida de mayor peso, que la mayoría de sus asociados ya estarán recuperados y así se ahorra significativamente en el espacio de almacenamiento de enlaces (el descodificador puede recuperar el espacio usado por los enlaces de entrada recuperados, y los símbolos de salida que están siendo procesados tendrán pocos asociados no recuperados aún).

El procesamiento de símbolos de salida en lotes de tamaños adecuados de peso creciente reduce los requisitos de memoria, así como los requisitos de procesamiento.

Según se muestra en la Fig. 15, se permite llegar a poco más que K símbolos de salida (indicados por K + A símbolos de salida en la figura) antes de que comience cualquier procesamiento (1515). Aquí, suponemos un símbolo de salida por paquete, los símbolos de entrada y de salida del mismo tamaño y K símbolos de entrada en el fichero de entrada. Inicialmente, el descodificador sencillamente espera la recepción de los K + A símbolos de salida, dado que el

descodificador no debería esperar poder recuperar el fichero de entrada a partir de menos de K + A símbolos de salida en cualquier caso, y no puede posiblemente recuperar un fichero de entrada arbitrario a partir de menos de K símbolos de salida. En la práctica, se halló que 5.√K es un buen valor para A.

Los números de filas en la OSDS 505 de símbolos de salida recibidos son almacenados en la tabla CLASIFICACIÓN DE PESOS 1405 de la Fig. 14 en orden de peso creciente, según se muestra en la etapa 1515 de la Fig. 15. Si T es el número de posibles pesos de símbolos de salida, entonces, para valores de L entre I y T, la lista LISTA_FILAS(L) contiene todos los símbolos de salida recibidos de peso VAL_WT(L), donde I=VAL_WT(1) < VAL_WT(2) < VAL_WT(3) < ... < VAL_WT(T) y VAL_WT(T) es el peso máximo de cualquier símbolo de salida. Luego, el resto del funcionamiento del descodificador mostrado en la Fig. 15 es exactamente el mismo que para el descodificador mostrado en la Fig. 13, excepto porque los símbolos de salida son procesados en orden de peso creciente, según se muestra en la etapa 1520.

Normalmente, K + A símbolos de salida bastarán para recuperar todos los símbolos de entrada Sin embargo, algunos conjuntos de K + A paquetes podrían no bastar para recuperar todos los símbolos de entrada. En tales casos, lotes de G paquetes adicionales son recibidos y luego procesados hasta que todos los símbolos de entrada sean recuperados. Un buen valor para G es √K.

Las Figs. 16 a 19 muestran una instantánea de una ejecución ejemplar del proceso descrito en la Fig. 15. En este ejemplo, seis símbolos de salida (16030, 16035, 16040, 16045, 16050, 16055) han sido recibidos, con asociados (16000, 16005, 16010, 16015, 16020, 16025) indicados según lo mostrado por las líneas con flechas en la Fig. 16 Inicialmente, los símbolos de salida con valores A, D, A⊕B⊕D⊕F, C, E⊕F y A⊕B (siendo la operación "⊕" una operación XOR) son recibidos y almacenados en la OSDS 505, según se muestra en la Fig. 17. El número de fila en la OSDS 505 es almacenado en la LISTA_FILAS en la fila correspondiente al peso del símbolo de salida, según se muestra en la Fig. 18. Los símbolos de salida de peso uno están en la fila 0, en la fila 1 y en la fila 3 de la OSDS 505. Así, LISTA_FILAS(0), que corresponde a símbolos de salida de peso VAL_WT(0) = 1, contiene los números de fila 0, 1 y 3, según se muestra en la Fig. 18. De manera similar, LISTA_FILAS(1) contiene 4 y 5 y LISTA_FILAS(3) contiene 2.

En este momento en el proceso, los primeros cinco símbolos de salida, en orden de peso creciente, han sido procesados, el sexto símbolo de salida en la fila 2 de la OSDS 505 ha sido procesado por el organizador 520 de recepción y este sexto símbolo de salida está por ser procesado por el procesador 525 de recuperación. Los símbolos de salida en las filas 0, 1, 3 y 5 ya han sido añadidos a la planificación para recuperar eventualmente símbolos de entrada en las posiciones 0, 3, 2 y 1, respectivamente. El símbolo de salida en la fila 4 de la OSDS 505 tiene dos asociados en las posiciones 4 y 5 que no han sido recuperados de momento, y por tanto hay enlaces desde las posiciones 4 y 5 en la ISDS 510, de vuelta a la fila 4 en la OSDS 505. El símbolo de salida en la fila 2 de la OSDS 505 tiene cuatro asociados en las posiciones 0, 1, 3 y 5. Los tres asociados en las posiciones 0, 1 y 3 ya han sido marcados como recuperados, y por tanto no hay ningún enlace desde ellos de vuelta a la fila 2 (causaron que el peso de este símbolo de salida se redujera de 4 a 1, lo que activará la recuperación de los restantes símbolos de entrada en las posiciones 4 y 5 una vez que el procesador 525 se ejecute). El asociado en la posición 5 no ha sido recuperado, y por tanto el organizador 520 de recepción añadió un enlace desde la posición 5 en la ISDS 510 a la fila 2 en la OSDS 505. Esto se muestra todo en la Fig. 17. Así, en este momento en el proceso, están en uso un total de solamente tres enlaces desde símbolos de entrada de vuelta a símbolos de salida que los tengan como asociados. Esto se compara favorablemente con la implementación directa que usa un enlace desde cada símbolo de entrada a cada símbolo de salida que lo tenga como un asociado. En este ejemplo, hay once enlaces posibles de ese tipo.

En general, los ahorros en el espacio de almacenamiento para los enlaces se reducen drásticamente al usar el proceso descrito en las Figs. 14 y 15 en lugar del proceso descrito en la Fig. 13, p. ej., los ahorros en el espacio son habitualmente un factor de 10 a 15 en el espacio de enlaces, cuando el número de símbolos de entrada es 50.000. La razón para esta reducción es que es más probable que los símbolos de salida de peso menor recuperen símbolos de entrada al comienzo del proceso que al final, y es mucho más probable que los símbolos de salida de pesos mayores recuperen símbolos de salida al final del proceso que al comienzo. Así, tiene sentido procesar los símbolos de salida en orden de peso creciente. Una ventaja adicional del proceso descrito en las Figs. 14 y 15 sobre la Fig. 13 es que la descodificación es habitualmente entre un 30% y un 40% más rápida. Esto es porque es más probable que los símbolos de salida de menor peso sean usados para recuperar símbolos de entrada, antes que los símbolos de salida de mayor peso (dado que los símbolos de salida de menor peso son considerados primero), y el coste de recuperar un símbolo de entrada específico depende directamente del peso del símbolo de salida usado para recuperarlo.

Selección de una distribución de pesos

Una optimización importante es diseñar el proceso de codificación de modo que un fichero de entrada pueda ser totalmente reconstruido con tan pocos símbolos de salida como sea posible. Esta optimización es útil allí donde el tiempo de transmisión y el ancho de banda son costosos o limitados, o allí donde un fichero de entrada debe ser descodificado rápidamente, sin esperar símbolos de salida adicionales. Habitualmente, el número suficiente de símbolos de salida necesarios para reconstruir un fichero de entrada es levemente mayor que el número de símbolos de entrada en el fichero

de entrada original (suponiendo símbolos de entrada y de salida del mismo tamaño). Puede mostrarse que un fichero de entrada arbitrario no puede ser recuperado cuando han sido recibidos menos bits que los que están en el fichero de entrada. Por lo tanto, un esquema perfecto de codificación permitirá la recuperación de un fichero de entrada a partir de cualquier conjunto de símbolos de salida, codificando el mismo número de bits que el fichero de entrada, y una medida de la eficacia de la codificación es cuán pocos bits extra se necesitan en condiciones esperadas.

En el descodificador mostrado en la Fig. 5, la máxima eficacia se obtiene cuando el procesador 525 de recuperación recupera el último símbolo de entrada desconocido después de que el descodificador ha recibido exactamente K símbolos de salida. Si más de K símbolos de salida han sido recibidos por el descodificador en el momento en que todos los símbolos de entrada pueden ser recuperados, entonces se habrían recibido símbolos de salida que no fueron necesarios, o usados, para la recuperación del fichero de entrada. Si bien la eficacia máxima es bonita, su búsqueda debería ser matizada por el riesgo de que la DSS 515 esté vacía antes de que la reconstrucción esté completa. En otras palabras, con máxima eficacia, el tamaño del conjunto descodificable llega a cero en cuanto acaba la reconstrucción, pero la codificación / descodificación debería disponerse de modo que no haya más que una pequeña probabilidad de que el tamaño del conjunto descodificable llegue a cero antes del final de la reconstrucción, usando K + A símbolos de salida, de modo que no se necesiten conjuntos adicionales de G símbolos de salida.

Esta cuestión se ilustra en la Fig. 20. Esa figura muestra un gráfico del tamaño de un conjunto descodificable con respecto al número de símbolos de entrada reconstruidos allí donde el descodificador está funcionando con K + A símbolos de salida para una distribución ideal descrita más adelante. En este ejemplo, A=0, es decir, el número de símbolos de salida recibidos para descodificar todos los K símbolos de entrada es el mínimo número posible (suponiendo que los símbolos de entrada y de salida tengan el mismo tamaño). Debería entenderse que el gráfico puede variar para cualquier función dada, para determinar pesos y asociados, y también variaría según cuáles símbolos de salida específicos sean recibidos. En ese gráfico, el tamaño esperado del conjunto descodificable es uno al principio y sigue siendo uno a lo largo del proceso de recuperación. Así, en el comportamiento esperado, siempre hay un símbolo de salida en el conjunto descodificable que puede ser usado para recuperar el próximo símbolo de entrada. La Fig. 20 también muestra un ejemplo del comportamiento efectivo de la distribución ideal. Obsérvese que en esta ejecución efectiva, el conjunto descodificable está vacío antes de que se complete la recuperación. Este comportamiento efectivo de la distribución ideal es típico, es decir, para la distribución ideal, las fluctuaciones aleatorias casi siempre vacían el conjunto descodificable antes de que se recuperen todos los símbolos de entrada, y esto es la razón por la cual se necesita una distribución más robusta, según se describe más adelante.

La eficacia mejora limitando el número de veces que un miembro del conjunto descodificable, que no tiene más que un símbolo de entrada asociado en el caso de una función de valor XOR, tiene un símbolo de entrada ya reconstruido como su asociado. Esto puede lograrse por una selección adecuada de la función para generar W(I).

Así, mientras es posible recuperar completamente un fichero de entrada con cualquier grado deseado de certidumbre, recibiendo suficientes símbolos de entrada, es preferible diseñar un sistema de comunicaciones de codificación por reacción en cadena, de modo que haya una alta probabilidad de recuperar los K símbolos de entrada que comprenden un fichero de entrada completo, con no más de K + A símbolos de salida (suponiendo el mismo tamaño para símbolos de entrada y símbolos de salida) para algún valor pequeño de A. El valor mínimo para A es cero, y puede ser logrado en algunos esquemas de codificación, tales como la codificación de Reed-Solomon, pero al aceptar algún valor pequeño, no nulo, para A, puede obtenerse un sistema de comunicaciones mejorado.

Los valores pequeños de A pueden ser logrados en la codificación por reacción en cadena, usando las distribuciones adecuadas que determinan la distribución de pesos para símbolos de salida, es decir, la distribución de W(I) sobre todo I, y la distribución de asociados sobre los símbolos de salida, es decir, los miembros de AL(I) sobre todo I. Debería subrayarse que, si bien el proceso de descodificación puede aplicarse independientemente de la distribución de pesos y de la distribución de la selección de los asociados, las realizaciones preferidas usarán distribuciones de pesos y distribuciones de la selección de los asociados específicamente escogidas para prestaciones casi óptimas. De hecho, muchas distribuciones funcionarán bien, ya que pequeñas variaciones en la distribución escogida pueden llevar a solamente pequeños cambios en las prestaciones.

La metodología para determinar las distribuciones en una realización preferida se describirán ahora. Las distribuciones de pesos efectivas usadas se basan en una distribución matemática ideal. En la distribución de pesos ideal, los pesos W(I) son escogidos de acuerdo a una distribución “ideal” de probabilidades. El peso más pequeño es uno y el peso más grande es K, donde K es el número de símbolos de entrada. En la distribución ideal, un peso igual a un valor de *i* es escogido con la siguiente probabilidad p:

$$\text{para } i=1: \qquad p=1/K; \text{ y}$$

$$\text{para } i=2,\ldots,K: \qquad p=1/(i(i-1)).$$

Una vez que un peso W(I) ha sido escogido, una lista AL(I) de W(I) símbolos de entrada asociados es escogida,

independientemente y uniformemente al azar (o seudo-aleatoriamente, si es necesario), asegurándose de que todos los asociados escogidos sean distintos. Así, el primer asociado es seleccionado aleatoriamente entre los K símbolos de entrada, teniendo cada uno una probabilidad de 1/K de ser seleccionado. El segundo asociado (si W>1) es luego seleccionado aleatoriamente entre los restantes K-1 símbolos. La distribución de probabilidades de pesos mostrada anteriormente tiene la propiedad de que, si el sistema se comportara exactamente como se espera, exactamente K símbolos de salida serían suficientes para descodificar y recuperar todos los símbolos de entrada. Este comportamiento esperado para la distribución ideal se muestra en la Fig. 20 con la línea continua. Sin embargo, debido a la naturaleza aleatoria de la selección de los pesos y los asociados, y porque un conjunto arbitrario de símbolos de salida es usado en el proceso de descodificación, el proceso no siempre se comportará de esa manera. Un ejemplo de un comportamiento efectivo para la distribución ideal se muestra en la Fig. 20 con la línea discontinua. Por tanto, la distribución ideal de pesos debe ser modificada en algún grado en la práctica.

Generalmente, los mejores parámetros para una configuración dada pueden ser hallados por simulación por ordenador. Sin embargo, una variación sencilla de la distribución de pesos ideal es una opción efectiva. En esta variación sencilla, la distribución de pesos ideal es modificada levemente aumentando la probabilidad de símbolos de salida de peso uno y de símbolos de salida de alto peso, para reducir la posibilidad de que el conjunto descodificable se vacíe antes de que los K + A símbolos de salida sean procesados. El suministro extra de símbolos de salida de peso uno reduce la posibilidad de que el proceso se quede sin símbolos de salida de peso uno (es decir, vacíe el conjunto descodificable) hasta que el proceso de recuperación esté cerca del final de la recuperación de los símbolos de entrada. El suministro extra de símbolos de salida de alto peso aumenta la posibilidad de que, cerca del final del proceso de recuperación, para cada símbolo de entrada no recuperado de momento, haya al menos un símbolo de salida que tenga ese símbolo de entrada como su único asociado restante.

Más específicamente, la distribución de pesos modificada es la siguiente:

para i=1: $p = n \cdot R1/K$;

para i=2,...,(K/R2 – 1): $p = n/(i(i-1)(1-iR2/K))$; y

para i=K/R2,...,K: $p = n \cdot HW(i)$

donde K es el número de símbolos de entrada, R1 y R2 son parámetros afinables y n es un factor de normalización usado de modo que todos los valores de p tengan suma igual a uno.

El cálculo de HW(*i*) y los valores de muestra para R1 y R2 se muestran en detalle en el Apéndice A. Allí, los símbolos de C++ nStartRippleSize, nRippleTargetSize y nSegments corresponden a R1, R2 y K, respectivamente, en las ecuaciones anteriores.

Esta distribución modificada es similar a la distribución matemática ideal de pesos, con más símbolos de salida de peso 1 y de mayor peso, y con la distribución reajustada en consecuencia. Según se muestra en la distribución modificada, R1 determina la fracción inicial de símbolos de salida de peso uno, así como determina el múltiplo de la probabilidad aumentada de símbolos de peso uno, y R2 determina el límite entre los pesos “más altos” y los pesos “no tan altos”.

Las buenas elecciones para los valores de R1 y R2 generalmente dependen de K y pueden ser determinadas empíricamente. Por ejemplo, hacer R1 igual a 1,4 veces la raíz cuadrada de K y R2 igual a dos más dos veces la raíz cuarta de K funciona bien en la práctica. Así, para K=4.000, fijar R1=89 y R2=18 funciona bien; cuando K es 64.000, fijar R1=354 y R2=34 funciona bien. Cálculos más detallados de R1 y R2 se muestran en el Apéndice A. La Fig. 21 muestra que el comportamiento esperado de esta distribución deja al conjunto descodificable moderadamente grande a lo largo del proceso de recuperación, de modo que, en ejecuciones efectivas, es improbable que las fluctuaciones aleatorias del comportamiento esperado vacíen el conjunto descodificable antes de que todos los símbolos de entrada sean recuperados.

Aunque los procesos de reconstrucción descritos anteriormente son similares al usado para códigos Tornado, el proceso distinto usado para construir el código lleva a efectos extremadamente distintos. En particular, según lo descrito anteriormente, la memoria requerida para la codificación por reacción en cadena es significativamente menos que para códigos Tornado, y la facilidad de uso de códigos de reacción en cadena en diversas situaciones supera ampliamente la de los códigos Tornado, posiblemente a cambio de algo menos de velocidad. Los detalles matemáticos que subyacen a los procesos se describen en más detalle más adelante.

Propiedades de algunos códigos de reacción en cadena

El número de símbolos de salida generados y enviados a través del canal no está limitado con la codificación por reacción en cadena, como ocurre con otros esquemas de codificación, dado que las claves no necesitan tener una correspondencia de uno a uno con los símbolos de entrada y el número de valores distintos de I no está limitado a alguna

razón de símbolos de entrada. Por lo tanto, es probable que, incluso si el conjunto descodificable se vacía antes de que el fichero de entrada esté reconstruido, el proceso de descodificación no fracasará, dado que el descodificador puede reunir tantos símbolos de salida más como sean necesarios para obtener al menos un símbolo de salida más de peso uno. Cuando ese símbolo de salida de peso uno es recibido, aumenta el conjunto descodificable y, por el efecto de reacción en cadena, podría causar la reducción de símbolos de salida anteriormente recibidos hasta el peso uno, de modo que ellos, a su vez, puedan ser usados para reconstruir símbolos de entrada.

En la mayoría de los ejemplos descritos anteriormente, los símbolos de entrada y salida se codifican para el mismo número de bits y cada símbolo de salida se coloca en un paquete (siendo un paquete una unidad de transporte que o bien es recibida en su totalidad o bien es perdida en su totalidad). En algunas realizaciones, el sistema de comunicaciones se modifica de modo que cada paquete contenga varios símbolos de salida. El tamaño de un valor de símbolo de salida se fija luego en un tamaño determinado por el tamaño de los valores de símbolos de entrada en la partición inicial del fichero en símbolos de entrada, en base a un cierto número de factores. El proceso de descodificación permanecería esencialmente sin cambios, excepto en que los símbolos de salida llegarían en manojos según fuera recibido cada paquete.

La determinación de los tamaños de símbolos de entrada y de símbolos de salida está usualmente dictada por el tamaño del fichero y el sistema de comunicación por el cual han de transmitirse los símbolos de salida. Por ejemplo, si un sistema de comunicación agrupa bits de datos en paquetes de un tamaño definido o agrupa bits de otras maneras, el diseño de los tamaños de símbolos comienza por el tamaño de paquete o de agrupación. A partir de allí, un diseñador determinaría cuántos símbolos de salida serán transportados en un paquete o grupo, y eso determina el tamaño del símbolo de salida. Para mayor simplicidad, el diseñador, probablemente, fijaría el tamaño del símbolo de entrada igual al tamaño del símbolo de salida, pero si los datos de entrada conforman un tamaño distinto del símbolo de entrada más conveniente, puede ser usado.

Otro factor al determinar el tamaño del símbolo de entrada es escoger el tamaño del símbolo de entrada de modo que el número de símbolos de entrada, K, sea lo bastante grande como para mantener mínimo el sobregasto de recepción. Por ejemplo, K=10.000 lleva a un sobregasto medio de recepción de entre el 5% y el 10% con fluctuaciones moderadas, mientras que K=80.000 lleva a un sobregasto medio de recepción de entre el 1% y el 2%, con muy poca fluctuación. Como ejemplo, en una prueba que comprende 1.000.000 de ensayos con K=80.000, el sobregasto de recepción nunca excedió el 4%.

El proceso de codificación descrito anteriormente produce un flujo de paquetes que contiene símbolos de salida basados en el fichero original. Los símbolos de salida contienen una forma codificada del fichero o, más sucintamente, el fichero codificado. Cada símbolo de salida en el flujo es generado independientemente de todos los otros símbolos de salida, y no hay ninguna cota inferior o superior sobre el número de símbolos de salida que pueden ser creados. Una clave está asociada a cada símbolo de salida. Esa clave, y algún contenido del fichero de entrada, determinan el valor del símbolo de salida. Los símbolos de salida generados consecutivamente no necesitan tener claves consecutivas y, en algunas aplicaciones, sería preferible generar aleatoriamente la secuencia de claves, o generar de manera seudo-aleatoria la secuencia.

La descodificación por reacción en cadena tiene la propiedad de que si el fichero original puede ser dividido en K símbolos de entrada de igual tamaño y cada valor de símbolo de salida tiene la misma longitud que un valor de símbolo de entrada, entonces el fichero puede ser recuperado a partir de K + A símbolos de salida en promedio, donde A es pequeño en comparación con K. Por ejemplo, A podría ser 500 para K=10.000. Dado que los símbolos de salida específicos son generados de una manera aleatoria o seudo-aleatoria, y la pérdida de símbolos de salida específicos en tránsito es arbitraria, existe alguna varianza pequeña en el número efectivo de símbolos de salida necesarios para recuperar el fichero de entrada. En algunos casos, allí donde una colección específica de K + A paquetes no es suficiente para descodificar el fichero de entrada entero, el fichero de entrada es aún recuperable si el receptor puede reunir más paquetes a partir de uno o más orígenes de paquetes de salida.

Debido a que el número de símbolos de salida está solamente limitado por la resolución de I, deberían poder ser generados bastantes más de K + A símbolos de salida. Por ejemplo, si I es un número de 32 bits, podrían ser generados 4 mil millones de símbolos de salida distintos, mientras que el fichero podría consistir en K=50.000 símbolos de entrada. En la práctica, solamente un pequeño número de esos 4 mil millones de símbolos de salida sería generado y transmitido, y es una casi certeza que un fichero de entrada puede ser recuperado con una fracción muy pequeña de los posibles símbolos de salida, y una excelente probabilidad de que el fichero de entrada pueda ser recuperado con poco más de K símbolos de salida (suponiendo que el tamaño del símbolo de entrada es el mismo que el tamaño del símbolo de salida).

El número medio de operaciones aritméticas requeridas para producir cada símbolo de salida es proporcional a log K y, por tanto, el número total de operaciones aritméticas requeridas para descodificar y recuperar el fichero de entrada es proporcional a K log K. Como se ha mostrado anteriormente, existe un proceso eficaz de descodificación que usa solo ligeramente más memoria que la memoria requerida para almacenar el fichero de entrada (habitualmente alrededor del

15% más). Los números anteriores muestran reducciones significativas en las operaciones y el almacenamiento, en comparación con técnicas de codificación previamente conocidas.

Por ejemplo, los códigos de Reed-Solomon son un código estándar para aplicaciones de comunicaciones. Con códigos de Reed-Solomon, el fichero de entrada es dividido en K símbolos de entrada, como con la codificación por reacción en cadena, pero los K símbolos de entrada en los códigos de Reed-Solomon son codificados en N símbolos de salida, donde N está habitualmente fijado antes de que comience el proceso de codificación. Esto contrasta con la presente invención, que admite un número indeterminado de símbolos de salida.

Una ventaja de tener un número indeterminado de símbolos de salida es que si un destinatario pierde más símbolos de salida de lo esperado, bien debido a un canal deficiente o bien debido a comenzar el destinatario después de que algunos símbolos de salida ya lo hayan dejado de lado, el destinatario puede simplemente escuchar un poco más y recoger más símbolos de salida. Otra ventaja es que, dado que el destinatario puede estar reuniendo símbolos de salida producidos desde múltiples codificadores, cada codificador puede tener que proporcionar solamente una pequeña fracción de los K símbolos de salida, y el número de símbolos provenientes de un codificador puede depender de cuántos codificadores están suministrando al destinatario los símbolos de salida.

Los códigos de Reed-Solomon también requieren significativamente más tiempo que los códigos por reacción en cadena, tanto para la codificación como para la descodificación. Por ejemplo, el número de operaciones aritméticas requeridas para producir cada símbolo de salida con Reed-Solomon es proporcional a K. El número de operaciones aritméticas requeridas para descodificar códigos de Reed-Solomon depende de cuáles símbolos de salida llegan al destinatario, pero generalmente el número de tales operaciones es proporcional a K^2. Por tanto, en la práctica, los valores aceptables de K y N son muy pequeños, del orden de las decenas y, posiblemente, hasta unos pocos cientos. Por ejemplo, los códigos Cruzados-Intercalados de Reed-Solomon son usados en discos compactos (CD) y CD-ROM. Para los CD, un código estándar usa K = 24 y N = 28 y otro código estándar usa K = 28 y N = 32. Para los CD-ROM, un código estándar usa K = 24 y N = 26 y otro código estándar usa K = 43 y N = 45. Los códigos estándar de Reed-Solomon usados para la transmisión satelital de ficheros MPEG (MPEG es un formato de fichero para flujos de vídeo y audio) usan K = 188 y N = 204; generalmente estos grandes valores requieren hardware especializado.

Se conoce la existencia de implementaciones más veloces de códigos de Reed-Solomon, que permiten la codificación en un tiempo cK *log* K y la descodificación en un tiempo c' K (*log* K)^2, pero c y c' son constantes prohibitivamente grandes que hacen estas implementaciones más lentas que otras implementaciones de códigos de Reed-Solomon, para todos los valores de K que no sean muy grandes, es decir, el punto de encrucijada de la eficacia es para valores de K en los miles o decenas de miles. Así, para valores de K por debajo del punto de encrucijada, las otras implementaciones de códigos de Reed-Solomon son más veloces. Aunque las implementaciones más veloces son más veloces que las otras implementaciones en valores de K por encima del punto de encrucijada, las implementaciones más veloces son más lentas en esos valores de K que los códigos por reacción en cadena, en algunos órdenes de magnitud.

Debido a las limitaciones de velocidad, solamente valores pequeños de K y N son generalmente viables para los códigos de Reed-Solomon. En consecuencia, su uso en grandes ficheros requiere que los ficheros sean divididos en muchos subficheros y que cada subfichero sea codificado por separado. Tal división reduce la efectividad de los códigos para protegerse ante la pérdida de paquetes en la transmisión.

Una característica de los códigos de Reed-Solomon es que K símbolos de salida distintos cualesquiera pueden ser usados por el destinatario para descodificar el fichero de entrada. Es demostrable que al menos K símbolos de salida son requeridos para descodificar un fichero de entrada arbitrario y, por tanto, los códigos de Reed-Solomon son óptimos en este aspecto, dado que K es también el máximo número de símbolos de salida necesarios para descodificar el fichero de entrada. La codificación por reacción en cadena, en cambio, requiere generalmente K + A paquetes, donde A es pequeño en comparación con un K adecuadamente escogido. En las aplicaciones de red descritas anteriormente, esta desventaja de necesitar, posiblemente, A símbolos adicionales, está ampliamente compensada por la ventaja en velocidad y la capacidad de manipular sin fisuras ficheros más grandes.

Variaciones del sistema básico de comunicación que realiza la presente invención

Los sistemas de comunicación para un único canal han sido descritos anteriormente en detalle. Los elementos de esas realizaciones pueden ser adaptados para usar ventajosamente más de un canal.

Las Figs. 22 a 23 muestran sistemas entre dos ordenadores que incorporan un sistema de comunicación tal como el mostrado en la Fig. 1. El primer ejemplo (Fig. 22) tiene un ordenador remitente 2200 enviando un fichero 2210 de entrada a un ordenador destinatario 2220 por una red 2230. El segundo ejemplo (Fig. 23) tiene un ordenador remitente 2300 difundiendo un fichero 2310 de entrada a ordenadores destinatarios 2320 (solamente se muestra uno) por un canal inalámbrico 2330. En lugar de la red 2330, podría usarse cualquier otro medio físico de comunicaciones, tal como los cables de Internet. El canal inalámbrico 2330 podría ser un canal de radio inalámbrico, un enlace de localizador, un enlace de satélite, un enlace infrarrojo, o similares. La configuración mostrada en la Fig. 23 podría también ser usada con medios

cableados o no cableados cuando un remitente está enviando un fichero de entrada a muchos destinatarios, cuando el destinatario está obteniendo el fichero de entrada desde muchos remitentes, o cuando muchos destinatarios están obteniendo el fichero de entrada desde muchos remitentes.

Como debería ser evidente al leer la descripción anterior, el esquema de codificación por reacción en cadena, descrito anteriormente, puede ser usado para enviar un fichero por un medio de transmisión con pérdidas, tal como una red de ordenadores, Internet, una red inalámbrica móvil o una red satelital, como un flujo de datos paquetizados con propiedades deseables. Una tal propiedad deseable es que, una vez que un agente descodificador recibe cualquier conjunto de suficientemente muchos paquetes desde el flujo, puede reconstruir el fichero original de manera extremadamente rápida. La codificación por reacción en cadena también es útil en situaciones donde están implicados muchos agentes, tal como cuando un agente transmisor está enviando el fichero a múltiples agentes receptores en una configuración de multidifusión o difusión.

El esquema de codificación por reacción en cadena también es útil en situaciones donde múltiples agentes transmisores están enviando el mismo fichero a múltiples agentes receptores. Esto admite una robustez mejorada ante el fallo localizado de la infraestructura de red, permitiendo a los agentes receptores cambiar sin fisuras la recepción de paquetes desde un agente transmisor a otro, y permite a los agentes receptores acelerar su descarga recibiendo desde más de un agente transmisor al mismo tiempo.

En un aspecto, el proceso de codificación por reacción en cadena, descrito anteriormente, realiza el equivalente digital de una imagen holográfica, donde cada parte de una transmisión contiene una imagen del fichero transmitido. Si el fichero es un fichero de un megaocteto, un usuario puede efectuar una toma desde un flujo transmitido para obtener un megaocteto arbitrario de datos (más algún sobregasto extra) y descodificar el fichero original de un megaocteto a partir de ese megaocteto.

Debido a que la codificación por reacción en cadena funciona con una selección aleatoria de datos desde el flujo transmitido, las descargas no necesitan estar planificadas o ser coherentes. Consideremos las ventajas del vídeo a petición con la codificación por reacción en cadena. Un vídeo específico podría ser difundido como un flujo continuo por un canal específico, sin coordinación entre el receptor y el transmisor. El receptor simplemente sintoniza un canal difusor para un vídeo de interés y captura suficientes datos para reconstruir el vídeo original, sin tener que averiguar cuándo comenzó la transmisión o cómo obtener copias de partes perdidas de la difusión.

Estos conceptos están ilustrados en las Figs. 24 a 25. La Fig. 24 ilustra una disposición en la cual un receptor 2402 recibe datos desde tres transmisores 2404 (indicados individualmente “A”, “B” y “C”) por tres canales 2406. Esta disposición puede ser usada para triplicar el ancho de banda disponible para el receptor o para tratar con los transmisores que no estén disponibles el tiempo suficiente como para obtener un fichero entero desde cualquier transmisor. Según se indica, cada transmisor 2404 envía un flujo de valores, S(). Cada valor S() representa un símbolo B(I) de salida y una clave I, cuyo uso está explicado en lo anterior. Por ejemplo, el valor $S(A, n_A)$ es el “n_A-ésimo” símbolo de salida y la “n_A-ésima” clave en una secuencia de símbolos de salida generados en el transmisor 2402(A). La secuencia de claves desde un transmisor es preferiblemente distinta a la secuencia de claves desde los otros transmisores, de modo que los transmisores no estén duplicando esfuerzos. Esto está ilustrado en la Fig. 24 por el hecho de que la secuencia S() es una función del transmisor.

Obsérvese que los transmisores 2402 no necesitan estar sincronizados o coordinados a fin de no duplicar esfuerzos. De hecho, sin coordinación, es probable que cada transmisor esté en una ubicación distinta en su secuencia (es decir, $n_A \neq n_B \neq n_C$). Dado que una selección aleatoria de K + A símbolos de salida, tal vez con unos pocos manojos de G símbolos de salida extra, puede ser usada para recuperar un fichero de entrada, las transmisiones no coordinadas son aditivas, en lugar de duplicativas.

Esta “aditividad de información” está ilustrada en la Fig. 25. Allí, copias de un fichero 2502 de entrada son proporcionadas a una pluralidad de transmisores 2504 (dos de los cuales se muestran en la figura). Los transmisores 2504 transmiten independientemente símbolos de salida generados a partir del contenido del fichero 2502 de entrada por los canales 2506 a los receptores 2508. Si cada transmisor usa un conjunto distinto de claves para la generación de símbolos, es probable que los flujos sean independientes y aditivos (es decir, añaden al fondo común de información usado para recuperar símbolos de entrada) en lugar de duplicativos. Cada transmisor de los dos mostrados podría necesitar transmitir solamente (K + A) / 2 símbolos de salida antes de que el descodificador del receptor sea capaz de recuperar el fichero de entrada entero.

Usando dos receptores y dos transmisores, la cantidad total de información recibida por una unidad receptora 2510 puede ser de hasta cuatro veces la información disponible por un canal 2506. La cantidad de información podría ser menor que cuatro veces la información de canal único si, por ejemplo, los transmisores difunden los mismos datos a ambos receptores. En ese caso, la cantidad de información en la unidad receptora 2510 es al menos el doble y a menudo más, si los datos se pierden en cualquier canal. Obsérvese que, incluso si los transmisores difunden solamente una señal, pero

los receptores están a la vista en distintos momentos, hay una ventaja en tener más de un receptor escuchando a cada transmisor. En la Fig. 25, la unidad receptora 2510 realiza las funciones similares a las funciones del receptor 150, el descodificador 155, el regenerador 160 de claves y el reensamblador 163 de ficheros de entrada mostrados en la Fig. 1.

En algunas realizaciones, el fichero 2502 de entrada es codificado en un dispositivo informático con dos codificadores, de modo que el dispositivo informático pueda proporcionar una salida para un transmisor y otra salida para el otro transmisor. Otras variaciones de estos ejemplos deberían ser evidentes tras la revisión de esta revelación.

Ha de entenderse que el aparato de codificación y los procedimientos descritos en el presente documento también pueden ser usados en otras situaciones de comunicación y que no están limitados a redes de comunicaciones tales como Internet. Por ejemplo, la tecnología de discos compactos también usa borrados y códigos correctores de errores para afrontar el problema de discos arañados, y se beneficiaría del uso de códigos de reacción en cadena al almacenar información sobre los mismos. Como otro ejemplo, los sistemas de satélites pueden usar códigos de borrado a fin de equilibrar los requisitos de potencia con la transmisión, admitiendo deliberadamente más errores al reducir la potencia, y la codificación por reacción en cadena sería útil en esa aplicación. Además, los códigos de borrado han sido usados para desarrollar sistemas RAID (formaciones redundantes de discos independientes) para la fiabilidad del almacenamiento de información. La actual invención puede, por lo tanto, demostrar su utilidad en otras aplicaciones tales como los ejemplos anteriores, donde los códigos son usados para afrontar los problemas de datos potencialmente propensos a pérdidas, o erróneos.

En algunas realizaciones preferidas, se proporcionan conjuntos de instrucciones (o software) para realizar los procesos de comunicación descritos anteriormente a dos o más máquinas informáticas de propósito múltiple, que se comunican por un medio de comunicaciones posiblemente propenso a pérdidas. El número de máquinas puede oscilar entre un remitente y un destinatario hasta cualquier número de máquinas enviando y / o recibiendo. El medio de comunicaciones que conecta las máquinas puede ser cableado, óptico, inalámbrico, o similares. Los sistemas de comunicaciones anteriormente descritos tienen muchos usos, lo que debería ser evidente a partir de esta descripción.

APÉNDICE A. LISTADOS DE CÓDIGO FUENTE

1. DFSolitonDistribution.cpp

```
// DFSolitonDistribution.cpp:
//          implementación de la clase CDFSolitonDistribution.
//
// La distribución de probabilidades de los pesos de símbolos de salida se calcula en el constructor y luego
// es mencionado por el Descodificador de Claves cuando decide qué peso y almacén común están
// asociados a una clave.
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#include “DFSolitonDistribution.h”
#include <math.h>
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//  Construcción / Destrucción
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
CFDSolitonDistribution::CDFSolitonDistribution(
                    int nSegments,
                    int nRippleTargetSize,
                    int nStartRippleSize)
{
    m_nSegments = nSegments;
    // A menos que los valores de R y S estén especificados, usar las constantes de DFSolitonDistribution.h
    // con las fórmulas basadas en la cuarta raíz y la raíz cuadrada del número de segmentos.
    m_nRippleTargetSize = nRippleTargetSize;
    // Si un tamaño deseado de onda no está prefijado, calcularlo como la cuarta raíz del número de segmentos,
    // ajustado por un desplazamiento y un multiplicador.
    if (!m_nRippleTargetSize)
        m_nRippleTargetSize =
                int (knRAdd + kDRFactor * sqrt(sqrt(m_nSegments)));
    m_nStartRippleSize = nStartRippleSize;
    if (!m_nStartRippleSize)
        m_nStartRippleSize =
                int (kdSFactor * sqrt(m_nSegments));
    //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    // Calcular parámetros de la distribución Soliton:
    //
```

```
// Esta es la distribución robusta modificada con densidad menguada en los pesos N / R, 2N / R, ... N
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
m_nModulus = 0x1 << knPrecision;
// Para la variación anterior de la distribución Robusta,
// usar sencillamente -1 como el valor
m_nRobustKinds = m_nSegments / m_nRippleTargetSize – 2;
if (m_nRobustKinds < 1)                    m_nRobustKinds = 1;
if (m_nRobustKinds > m_nSegments)          m_nRobustKinds = m_nSegments;
// En la variación anterior de la distribución Robusta,
// m_nTailKinds es 0
m_nTailKinds = 1;
for (int d = m_nRippleTargetSize; d > 1; d /= 2)
    m_nTailKinds++;      // pasa a ser log_2 (RippleTargetSize) + 1
m_nKinds = m_nRobustKinds + m_nTailKinds;
if (m-nKinds > m_nSegments)
{
      m_nTailKinds = m_nSegments – m_nRobustKinds;
      m_nKinds = m_nSegments;
}
m_anKindWeight                = new int[m_nKinds];
m_anKindCumulativeDensity = new int[m_nKinds];
// adKindFraction es la distribución no normalizada
double* adKindFraction = new double [m_nKinds];
// Símbolos de salida de peso 1:
adKindFraction[0] = double (m_nStartRippleSize) / double (m_nSegments);
m_anKindWeight[0] = 1;
// Densidades según la distribución Soliton robusta:
for (int i=1, i < m_nRobustKinds; i++)
{
      adKindFraction[i] = adKindFraction[i-1] + 1.0 / (double(i) *
          double(i+1) * (1.0 – double ((i+1) * m_nRippleTargetSize)
                                        / double (m_nSegments)));
      m_anKindWeight[i] = i + 1;
}
int nRPower  = 1;
```

```
        int j;
        for (i = 0; i < m_nTailKinds; i++)
              nRPower *= 2; // nRPower es la siguiente potencia de 2 > m_nRippleTargetSize
        // Densidades para la cola menguada al final de la distribución.
        // Los pesos descienden desde m_nSegments en un factor de 2 cada vez
        // y la densidad es inversamente proporcional al peso.
        // j va desde 2 hasta nRPower
        for (i=m_nRobustKinds, j=2, i < m_nKinds; i++, j*=2)
        {
            adKindFraction[i] = adKindFraction[i-1]
                  + kdTFactor * double (nRPower) / double (j*m_nSegments);
            m_anKindWeight[i] = (j*m_nSegments) / nRPower;
        }
        // Normalizar a m_nModulus
        for (i = 0; i < m_nKinds; i++)
            m_anKindCumulativeDensity[i] = int(adKindFraction[i] *
                                    double (m_nModulus) / adKindFraction [m_nKinds – 1]);
            delete adKindFraction;
        // Calcular peso máximo y medio
        m_nMaxWeight = 0;
        for (i = 0; i < m_nKinds; i++)
            if (m_anKindWeight[i] > m_nMaxWeight)
                                    m_nMaxWeight = m_anKindWeight[i];
        ComputeAverageWeight();
    }
    CDFSolitonDistribution:: -CDFSolitonDistribution()
    {
        if (m_anKindWeight)                 delete [] m_anKindWeight;
        if (m_anKindCumulativeDensity)      delete [] m_anKindCumulativeDensity;
    }
    void CDFSolitonDistribution::ComputeAverageWeight()
    {
        int i;
        float fTemp;
        fTemp = float (m_anKindWeight[0]) *
```

```
        float (m_anKindCumulativeDensity[0]);
        for (i=1; i < m_nKinds; i++)
        fTemp += float (m_anKindWeight[i]) *
        float (m_anKindCumulativeDensity[i] -
        m_anKindCumulativeDensity[i-1])
        m_nAverageWeight = int((fTemp / m_nModulus) + 1);
    }
```

2. DFSolitonDistribution.h

```
// DFSolitonDistribution.h: interfaz para la clase CDFSolitonDistribution
//
// Nota: La clase CDFDominoKeyDecoder calcula el peso y el almacén común de los vecinos para una
// clave dada de símbolo de salida. Se refiere a esta clase para la distribución de probabilidades para
// los pesos y se refiere a la clase CDFRandomBits para los enteros aleatorios usados para hacer las
// selecciones.
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#include “DFRandomBits.h”
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//
// Constantes usadas en el cálculo de la distribución
//
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// La precisión aritmética de la distribución en bits.
// La densidad de la distribución está ajustada a la 2ª potencia de este valor.
const int knPrecision = 20;
//Constantes relacionadas con el cálculo de R, el “tamaño deseado de onda”.
// Este valor es el tamaño de onda esperado en la “Distribución Soliton Robusta”.
// R obtiene el valor kdRFactor * 4ª raíz de N + knRadd
// donde N es el número de símbolos de entrada (segmentos).
const double kdRFactor = 2.0;
const int knRadd = 2;
// S es el tamaño esperado de la onda al comienzo de la descodificación,
// con el significado del número esperado de símbolos de salida de peso 1 cuando son recibidos N símbolos
// S = kdSFactor * sqrt(N)
const double kdSFactor = 1.4;
// La cola de la distribución recibe mayor densidad en los pesos N, N/2,..., hasta N/R. La densidad de la
```

```
// distribución es inversamente proporcional al peso del símbolo, con kdTFactor como una constante de
// proporcionalidad. La distribución resultante está aún ajustada según la precisión anterior.
const double kdTFactor = 1.6;
class CDFSolitonDistribution
{
    friend class CDFDominoKeyDecoder;
public;
    ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    // Clave de símbolo de salida y posición de símbolo de entrada
    //
    // Esta clase define los tipos TKey y TPos, que vale por Posición de Símbolo de Entrada. Otras clases que
    // necesiten estos tipos deberían referirse a estos más adelante.
    ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    typedef unsigned int TKey;
    typedef int          TPos;
    CDFSolitonDistribution (
                int nSegments,
                int nRippleTargetSize = 0,
                int nStartRippleSize = 0);
    virtual –CDFSolitonDistribution();
    inline int nAverageWeight ()    {return m_nAverageWeight; };
    inline int nMaxWeight ()        {return m_nMaxWeight; };
    inline int nParameterR ()       {return m_nRippleTargetSize; };
    inline int nParameterS ()       {return m_nStartRippleSize; };
    // La distribución de probabilidades comprende una formación de clases donde
    // cada clase corresponde a un peso para símbolos de salida y una densidad (probabilidad) asociada a ese peso.
    inline int nKinds ()      { return m_nKinds; };
    private:
    int m_nAverageWeight;
    void ComputeAverageWeight ();
    int m_nRippleTargetSize;
    int m_nStartRippleSize;
    // m_nKinds es el tamaño de la formación que contiene la distribución de probabilidades. Es el número de
    // pesos distintos de símbolos de salida que son posibles.
    int m_nKinds;
```

```
// Los siguientes son el número de clases de la distribución Soliton robusta truncada, y el número de clases
// debido a la distribución de cola menguada.
int m_nRobustKinds;
int m_nTailKinds;
int m_nModulus; // 2 a la precisión
int* m_anKindWeight; // el peso correspondiente a una clase
int* m_anKindCumulativeDensity; // densidad de probabilidad de una clase
int m_nSegments;
int m_nMaxWeight;
```

REIVINDICACIONES

1. Un procedimiento de transmitir datos desde un origen a través de una pluralidad de canales, que comprende las etapas de:

a) disponer los datos a transmitir como un conjunto ordenado de símbolos de entrada;

b) generar una pluralidad de símbolos de salida para cada uno entre la pluralidad de canales, en donde cada símbolo de salida es generado por las etapas de:

1) seleccionar, de un alfabeto de claves, una clave I para el símbolo de salida que está siendo generado, en donde el alfabeto de claves contiene muchos más miembros que el número de símbolos de entrada en el conjunto ordenado de símbolos de entrada, de modo que pueda ser generado un número efectivamente ilimitado de símbolos de salida que sean generalmente independientes entre sí;

2) determinar un peso, W(I), como una función de I, en que los pesos W son enteros positivos que varían entre al menos dos valores y son mayores que uno para al menos algunas claves en el alfabeto de claves;

3) seleccionar W(I) de los símbolos de entrada según una función de I, formando así una lista AL(I) de W(I) símbolos de entrada asociados al símbolo de salida; y

4) calcular un valor B(I) del símbolo de salida a partir de una función de valor predeterminado de los W(I) símbolos de entrada asociados;

c) paquetizar al menos uno entre la pluralidad de símbolos de salida en cada uno entre una pluralidad de paquetes; y

d) transmitir la pluralidad de paquetes por la pluralidad de canales;

en el cual al menos dos entre la pluralidad de canales llevan paquetes que contienen símbolos de salida generados con valores distintos para la clave I, de modo que un destinatario pueda descodificar el conjunto ordenado de símbolos de entrada a partir de un cierto número de símbolos de salida igual a, o levemente mayor que, el número de símbolos de entrada en el conjunto ordenado de símbolos de entrada, suponiendo que los símbolos de entrada y los símbolos de salida representan el mismo número de bits de datos.

2. El procedimiento de la reivindicación 1, en el cual el origen comprende un único origen que genera valores para la clave I para el único origen y distribuye paquetes de salida por la pluralidad de canales, de modo que el contenido de información recibido desde dos cualesquiera entre la pluralidad de canales no sea enteramente duplicativo.

3. El procedimiento de la reivindicación 1, en el cual los valores para la clave I son seleccionados independientemente para cada uno entre la pluralidad de canales, de modo que el contenido de información recibido desde dos cualesquiera entre la pluralidad de canales no sea enteramente duplicativo.

4. El procedimiento de la reivindicación 1, en el cual la etapa de determinación comprende las etapas de:

calcular, según una función predeterminada de I y una distribución de probabilidades, dicho peso W(I), en donde la distribución de probabilidades es sobre al menos dos enteros positivos, al menos uno de los cuales es mayor que uno;

calcular una entrada de lista para la lista AL(I); y repetir la etapa de calcular una entrada de lista para la lista AL(I) hasta que se calculen W(I) entradas de la lista.

5. El procedimiento de la reivindicación 4, en el cual la etapa de determinar W(I) comprende una etapa de determinar W(I) de modo que W se aproxime a una distribución predeterminada sobre el alfabeto de claves.

6. El procedimiento de la reivindicación 5, en el cual la distribución predeterminada es una distribución uniforme.

7. El procedimiento de la reivindicación 5, en el cual la distribución predeterminada es una distribución de curva de campana.

8. El procedimiento de la reivindicación 5, en el cual la distribución predeterminada es tal que W=1 tiene una probabilidad de 1/K, donde K es el número de símbolos de entrada en el fichero de entrada, y W=i tiene una probabilidad de 1/i (i-1) para i=2,..., K.

9. El procedimiento de la reivindicación 5, en el cual la distribución predeterminada es tal que, dados parámetros R1 y R2 afinables, y siendo K el número de símbolos de entrada en el fichero de entrada, el peso W=1 tiene una probabilidad proporcional a R1/K, los pesos en una clase de peso bajo, que oscila entre el peso W=2 y el peso W=K / R2-1, tienen una probabilidad proporcional a 1 / (W (W-1) (1-W-R2 / K)) y los pesos en una clase de peso alto, que oscilan desde el peso

W=K / R2 hasta el peso W=K, tienen una distribución de probabilidad seleccionada.

10. Un aparato para transmitir datos desde un origen a través de una pluralidad de canales, que comprende:

a) medios para disponer los datos a transmitir como un conjunto ordenado de símbolos de entrada;

b) medios que generan una pluralidad de símbolos de salida para cada uno entre la pluralidad de canales, en donde dichos medios están configurados para generar cada símbolo de salida:

1) seleccionando, entre un alfabeto de claves, una clave I para el símbolo de salida que está siendo generado, en donde el alfabeto de claves contiene muchos más miembros que el número de símbolos de entrada en el conjunto ordenado de símbolos de entrada, de modo que pueda ser generado un número efectivamente ilimitado de símbolos de salida que sean generalmente independientes entre sí;

2) determinando un peso, W(I), como una función de I, en donde los pesos W son enteros positivos que varían entre al menos dos valores y son mayores que uno para al menos algunas claves en el alfabeto de claves;

3) seleccionando W(I) de los símbolos de entrada según una función de I, formando así una lista AL(I) de W(I) símbolos de entrada asociados al símbolo de salida; y

4) calculando un valor B(I) del símbolo de salida a partir de una función de valor predeterminado de los W(I) símbolos de entrada asociados;

c) medios para paquetizar al menos uno entre la pluralidad de símbolos de salida en cada uno entre una pluralidad de paquetes; y

d) medios para transmitir la pluralidad de paquetes por la pluralidad de canales, en donde al menos dos entre la pluralidad de canales llevan paquetes que contienen símbolos de salida generados con valores distintos para la clave I, de modo que un destinatario pueda descodificar el conjunto ordenado de símbolos de entrada a partir de un cierto número de símbolos de salida iguales a, o levemente mayores que, el número de símbolos de entrada en el conjunto ordenado de símbolos de entrada, suponiendo que los símbolos de entrada y los símbolos de salida representen el mismo número de bits de datos.

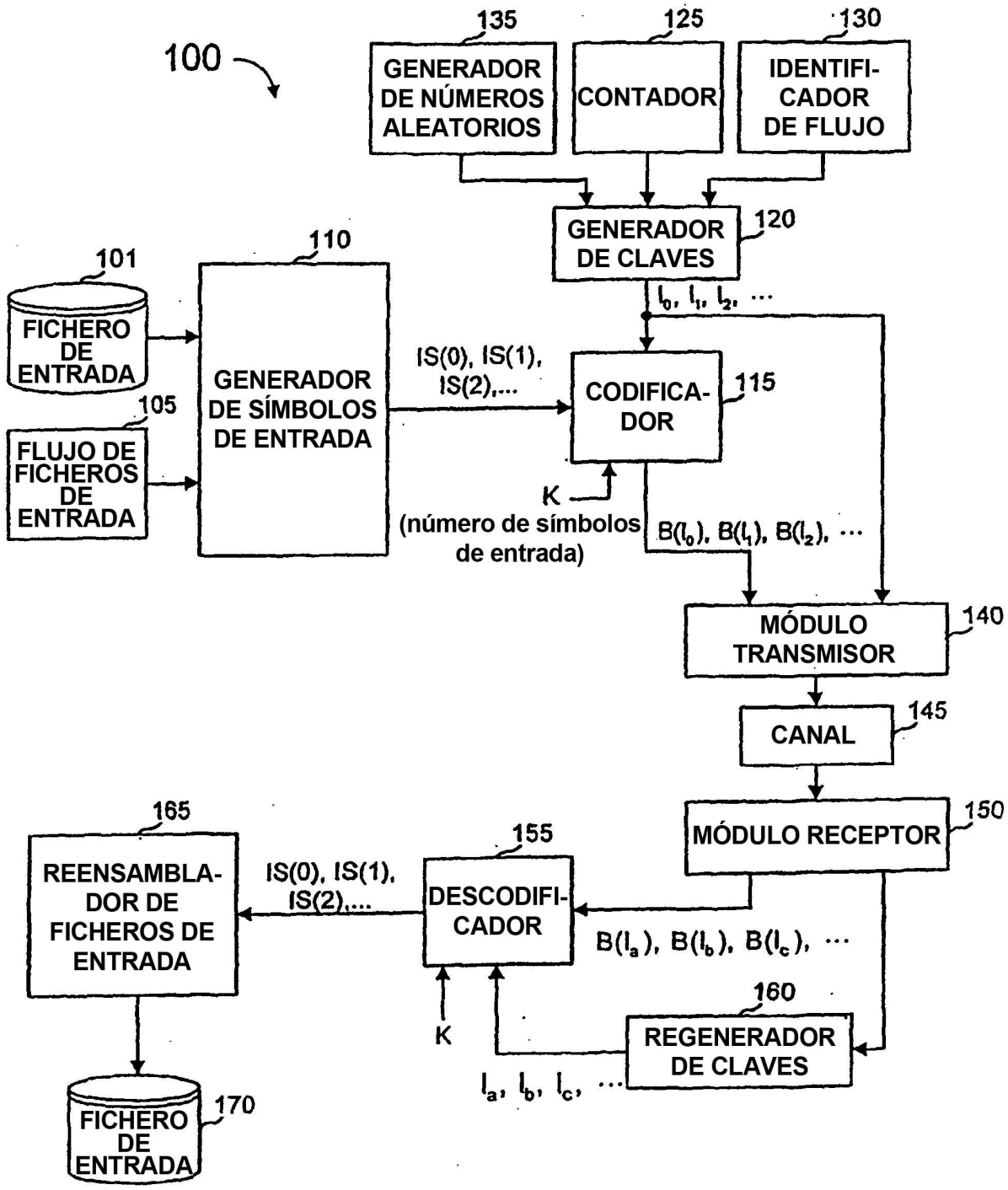
100
135
GENERADOR DE NÚMEROS ALEATORIOS
125
CONTADOR
130
IDENTIFI-CADOR DE FLUJO
GENERADOR DE CLAVES
120
$l_0, l_1, l_2, \cdots$
101
FICHERO DE ENTRADA
110
GENERADOR DE SÍMBOLOS DE ENTRADA
105
FLUJO DE FICHEROS DE ENTRADA
IS(0), IS(1), IS(2),...
CODIFICA-DOR
115
K
(número de símbolos de entrada)
$B(l_0), B(l_1), B(l_2), \cdots$
MÓDULO TRANSMISOR
140
CANAL
145
MÓDULO RECEPTOR
150
165
REENSAMBLA-DOR DE FICHEROS DE ENTRADA
IS(0), IS(1), IS(2),...
155
DESCODIFI-CADOR
$B(l_a), B(l_b), B(l_c), \cdots$
160
K
REGENERADOR DE CLAVES
$l_a, l_b, l_c, \cdots$
FICHERO DE ENTRADA
170

Figura 1

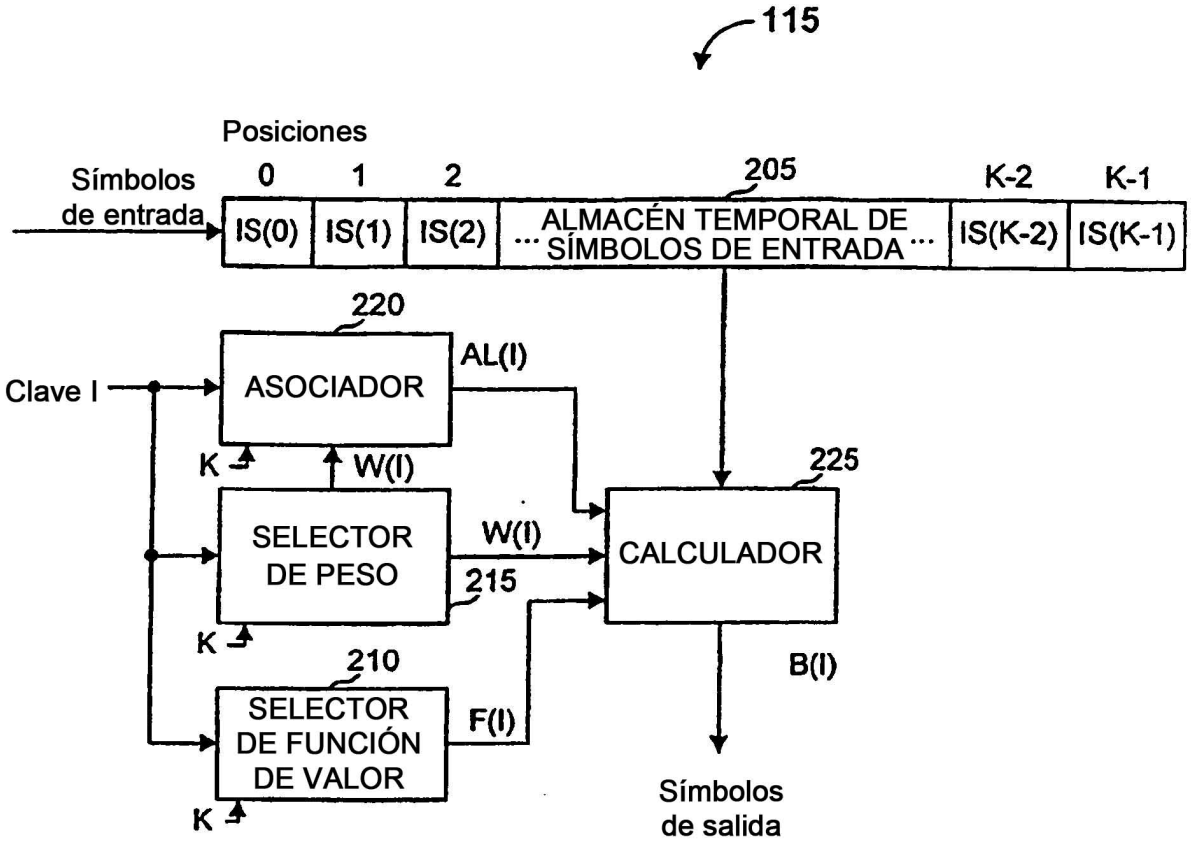

Figura 2

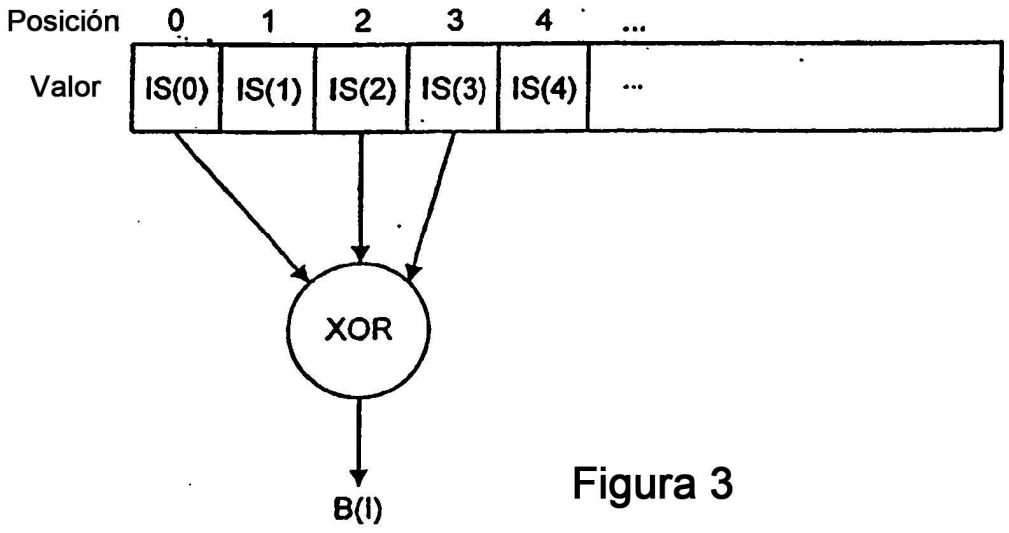

Figura 3

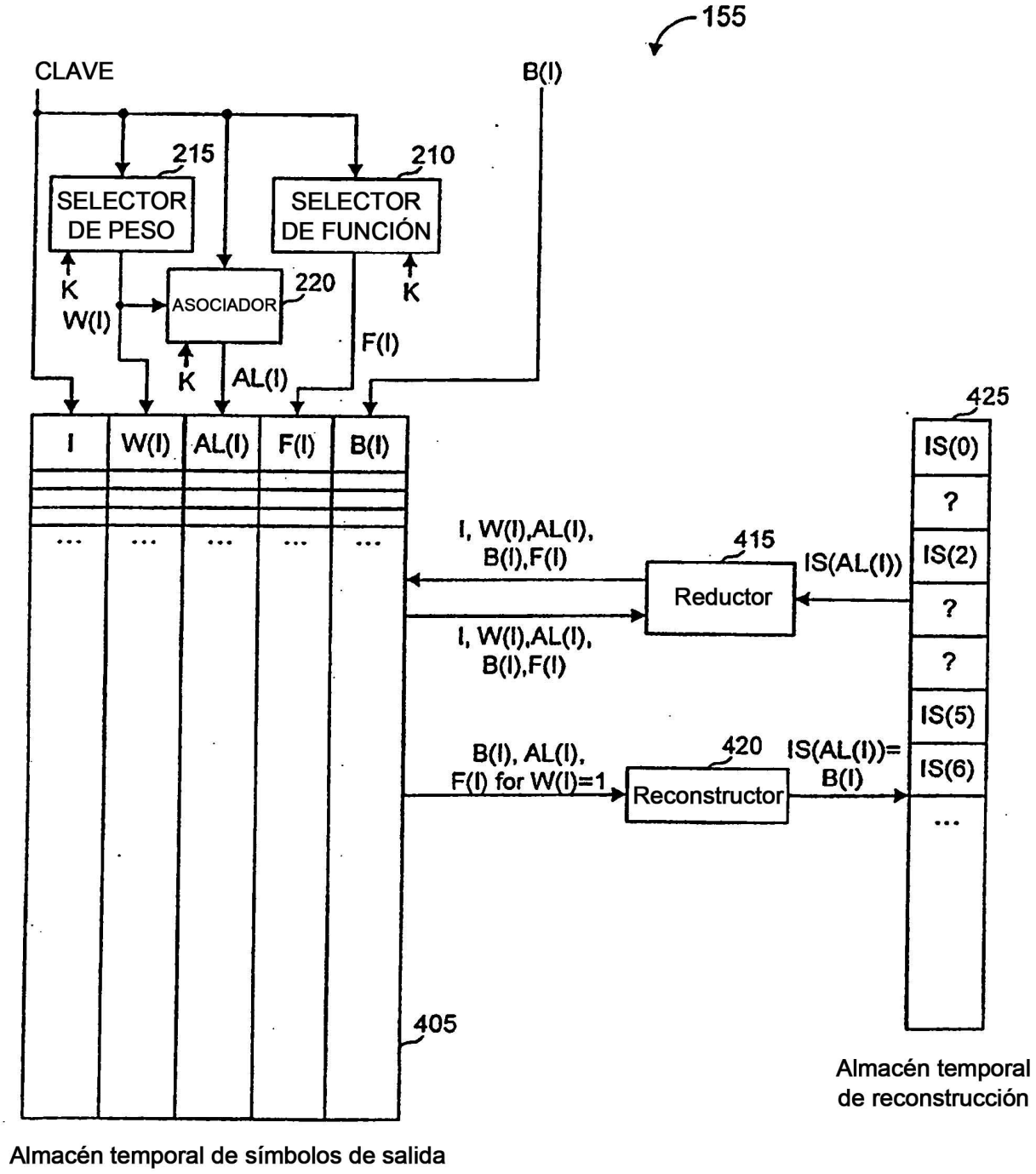
155
CLAVE
B(I)
215
SELECTOR DE PESO
210
SELECTOR DE FUNCIÓN
K
W(I)
ASOCIADOR
220
K
F(I)
K
AL(I)
I
W(I)
AL(I)
F(I)
B(I)
425
IS(0)
?
IS(2)
?
?
IS(5)
IS(6)
I, W(I),AL(I), B(I),F(I)
415
Reductor
IS(AL(I))
I, W(I),AL(I), B(I),F(I)
B(I), AL(I), F(I) for W(I)=1
420
Reconstructor
IS(AL(I))= B(I)
405
Almacén temporal de símbolos de salida
Almacén temporal de reconstrucción

Figura 4

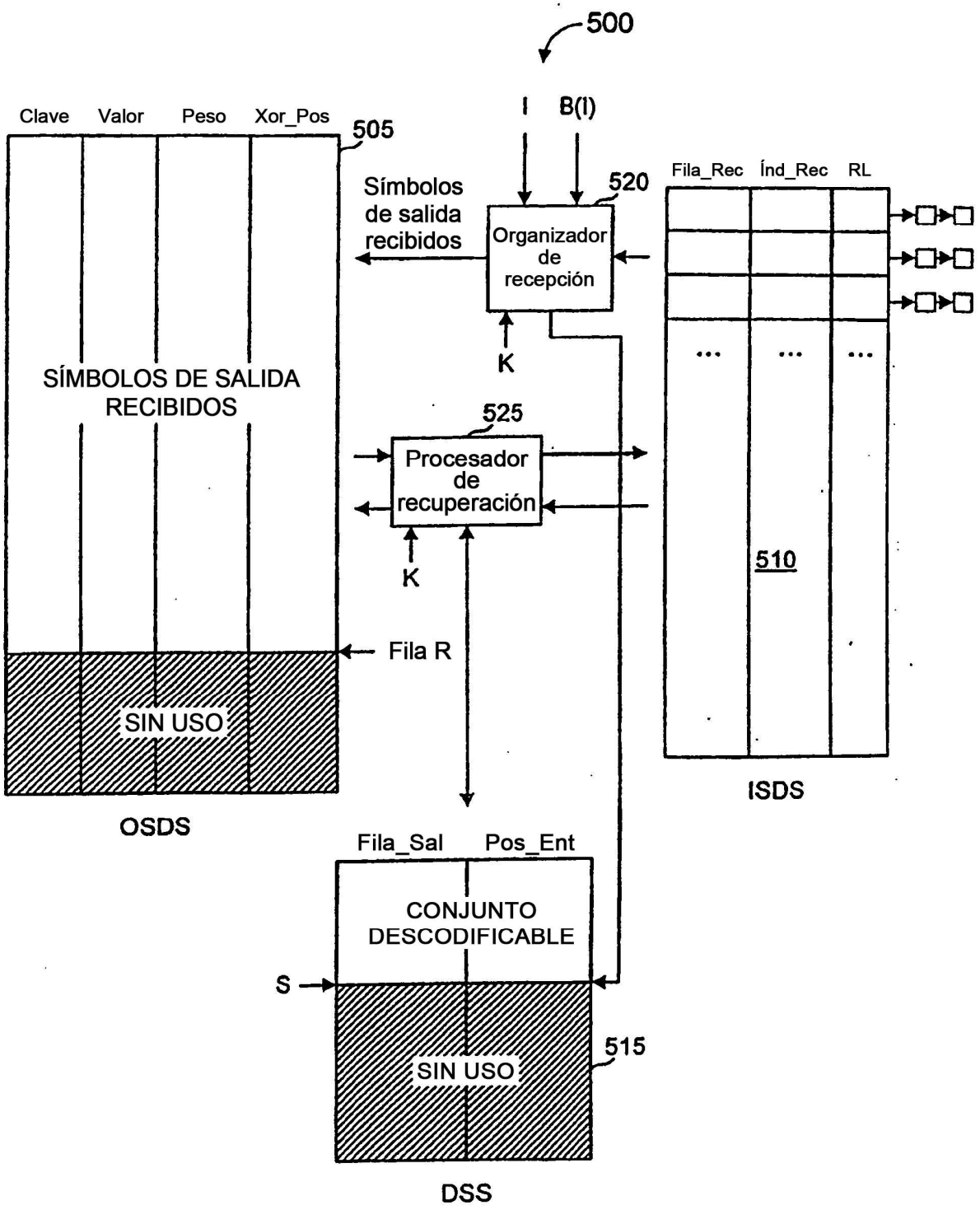
500
I
B(I)
Clave
Valor
Peso
Xor_Pos
505
Símbolos de salida recibidos
520
Organizador de recepción
Fila_Rec
Índ_Rec
RL
K
SÍMBOLOS DE SALIDA RECIBIDOS
525
Procesador de recuperación
K
510
Fila R
SIN USO
OSDS
ISDS
Fila_Sal
Pos_Ent
CONJUNTO DESCODIFICABLE
S
515
SIN USO
DSS

Figura 5

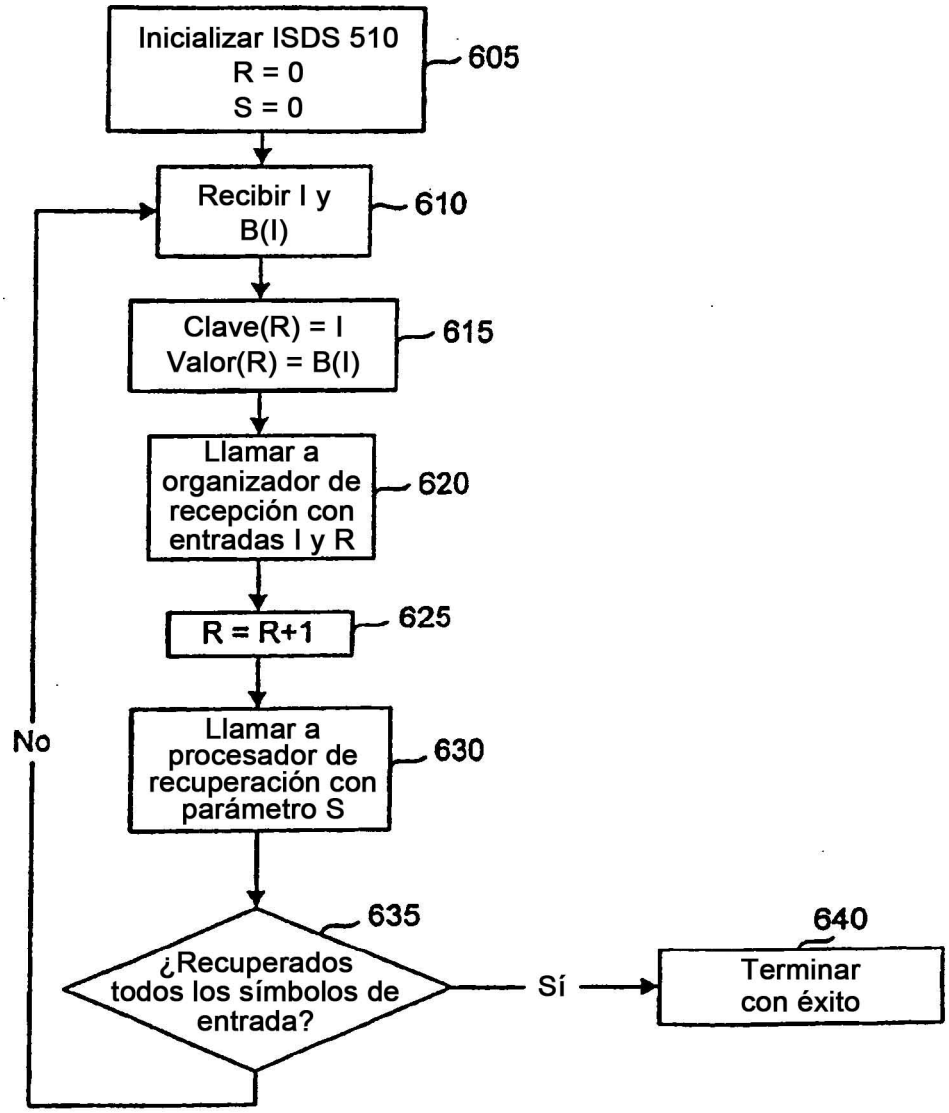
Inicializar ISDS 510
R = 0
S = 0
605
Recibir I y B(I)
610
Clave(R) = I
Valor(R) = B(I)
615
Llamar a organizador de recepción con entradas I y R
620
R = R+1
625
Llamar a procesador de recuperación con parámetro S
630
635
¿Recuperados todos los símbolos de entrada?
No
Sí
640
Terminar con éxito

Figura 6

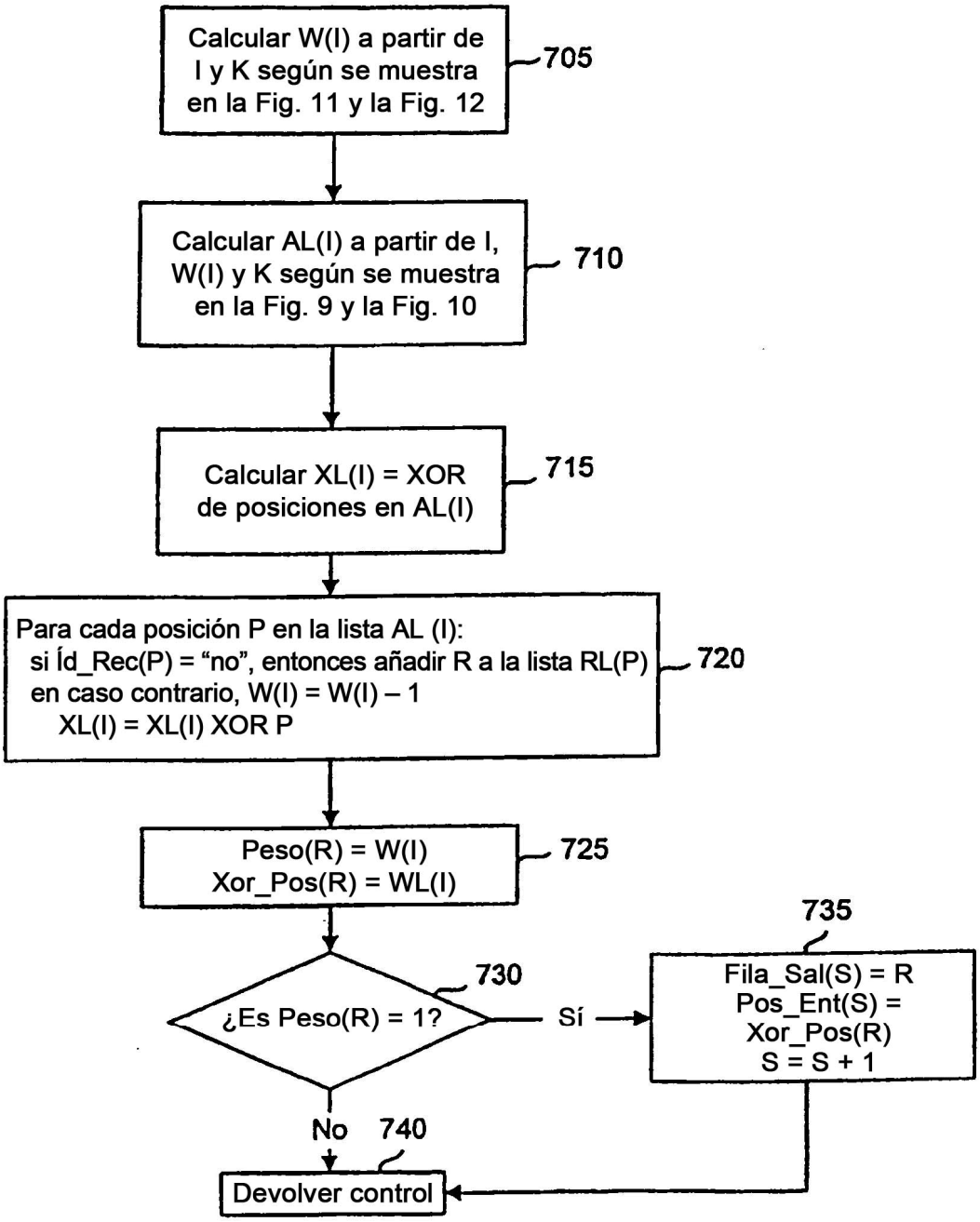
Calcular W(I) a partir de I y K según se muestra en la Fig. 11 y la Fig. 12
705
Calcular AL(I) a partir de I, W(I) y K según se muestra en la Fig. 9 y la Fig. 10
710
Calcular XL(I) = XOR de posiciones en AL(I)
715
Para cada posición P en la lista AL (I):
si Íd_Rec(P) = "no", entonces añadir R a la lista RL(P)
en caso contrario, W(I) = W(I) – 1
XL(I) = XL(I) XOR P
720
Peso(R) = W(I)
Xor_Pos(R) = WL(I)
725
735
730
¿Es Peso(R) = 1?
Sí
Fila_Sal(S) = R
Pos_Ent(S) = Xor_Pos(R)
S = S + 1
No
740
Devolver control

Figura 7

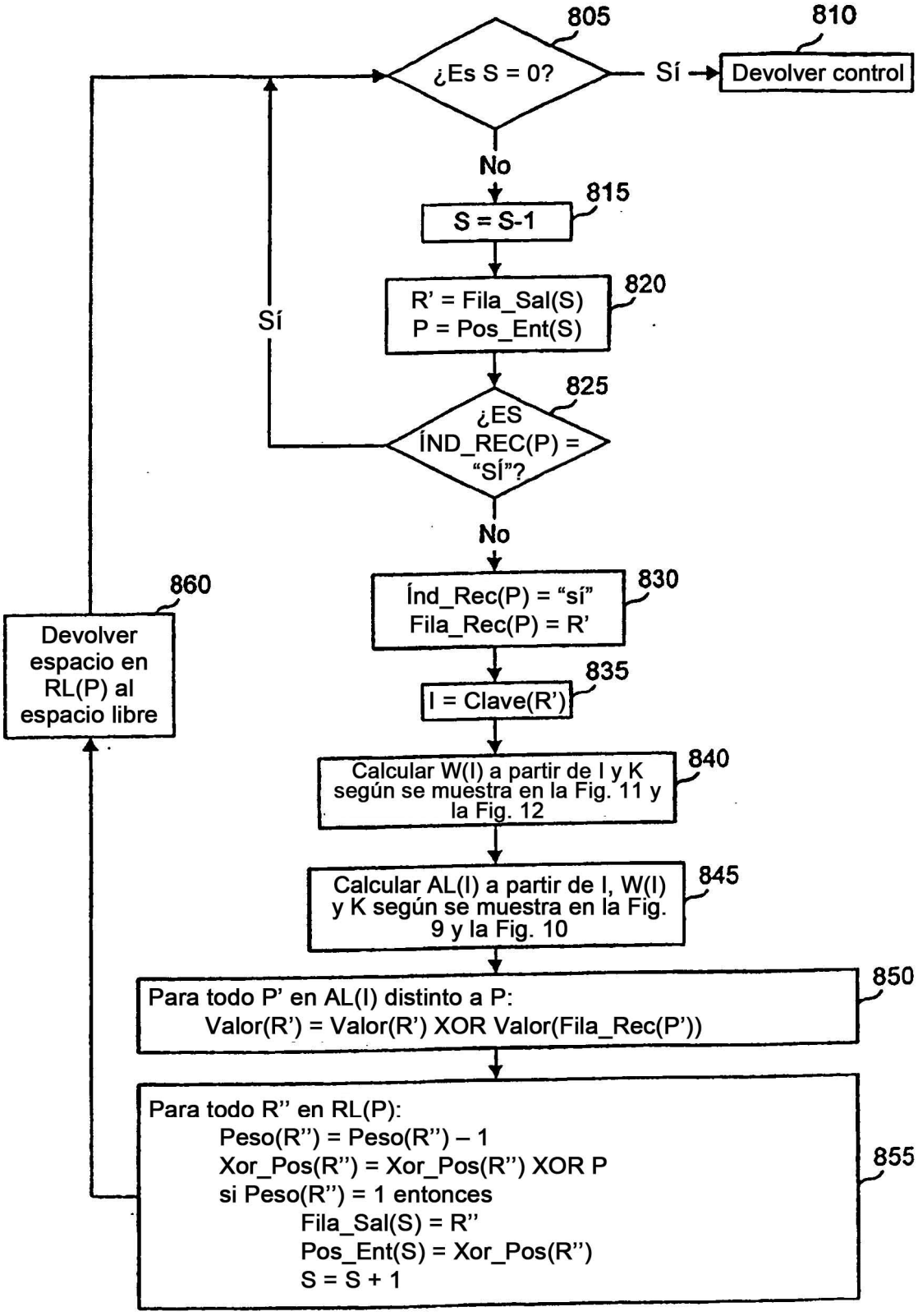
805
¿Es S = 0?
Sí
810
Devolver control
No
815
S = S-1
820
R' = Fila_Sal(S)
P = Pos_Ent(S)
Sí
825
¿ES ÍND_REC(P) = "SÍ"?
No
860
Devolver espacio en RL(P) al espacio libre
830
Índ_Rec(P) = "sí"
Fila_Rec(P) = R'
835
I = Clave(R')
840
Calcular W(I) a partir de I y K según se muestra en la Fig. 11 y la Fig. 12
845
Calcular AL(I) a partir de I, W(I) y K según se muestra en la Fig. 9 y la Fig. 10
850
Para todo P' en AL(I) distinto a P:
Valor(R') = Valor(R') XOR Valor(Fila_Rec(P'))
855
Para todo R'' en RL(P):
Peso(R'') = Peso(R'') – 1
Xor_Pos(R'') = Xor_Pos(R'') XOR P
si Peso(R'') = 1 entonces
Fila_Sal(S) = R''
Pos_Ent(S) = Xor_Pos(R'')
S = S + 1

Figura 8(a)

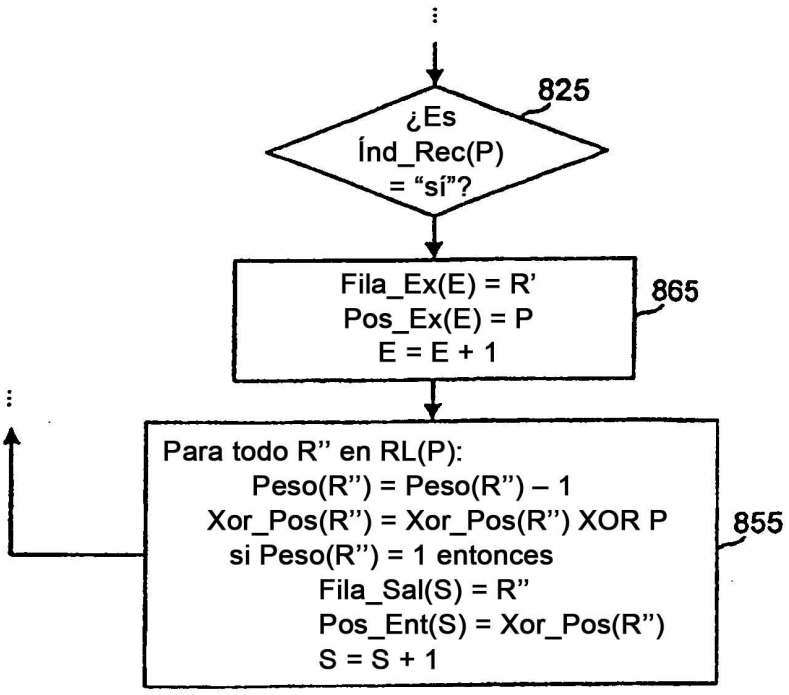

Figura 8(b)

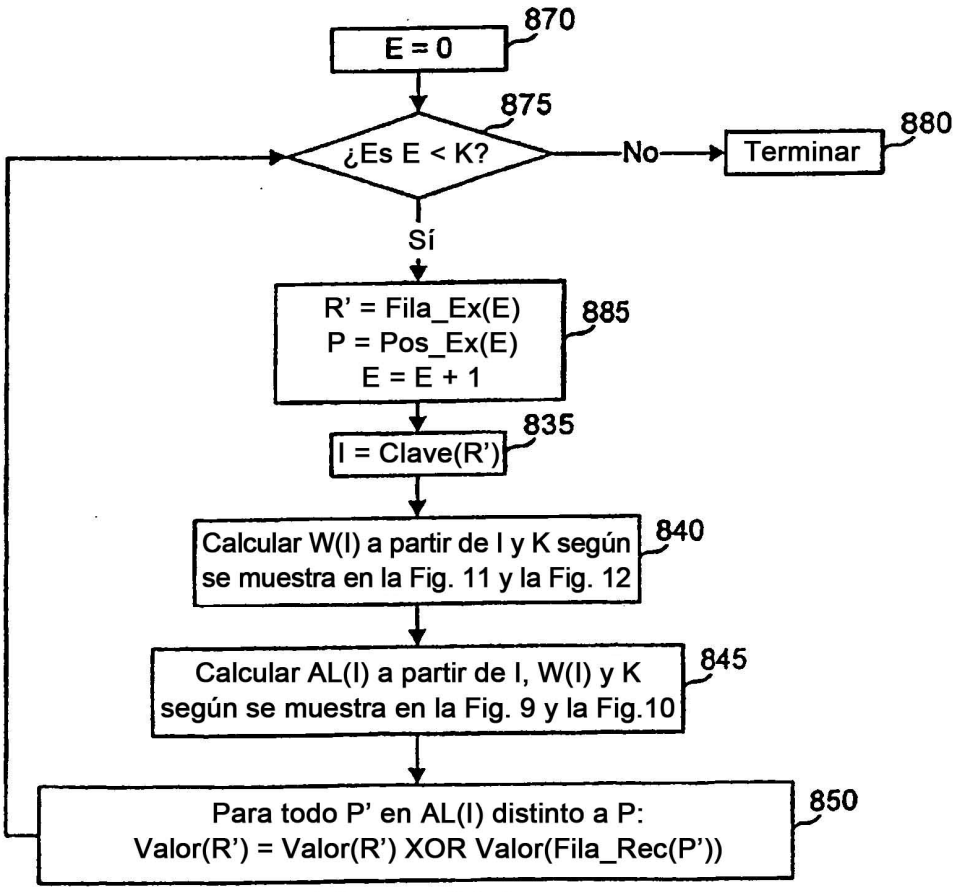

Figura 8(c)

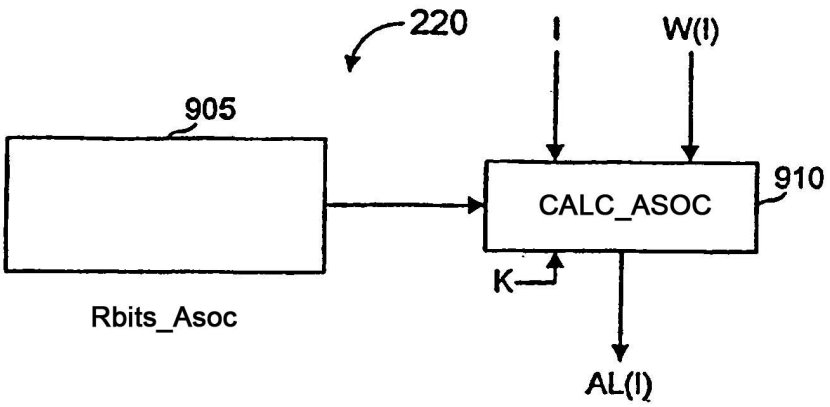

Figura 9

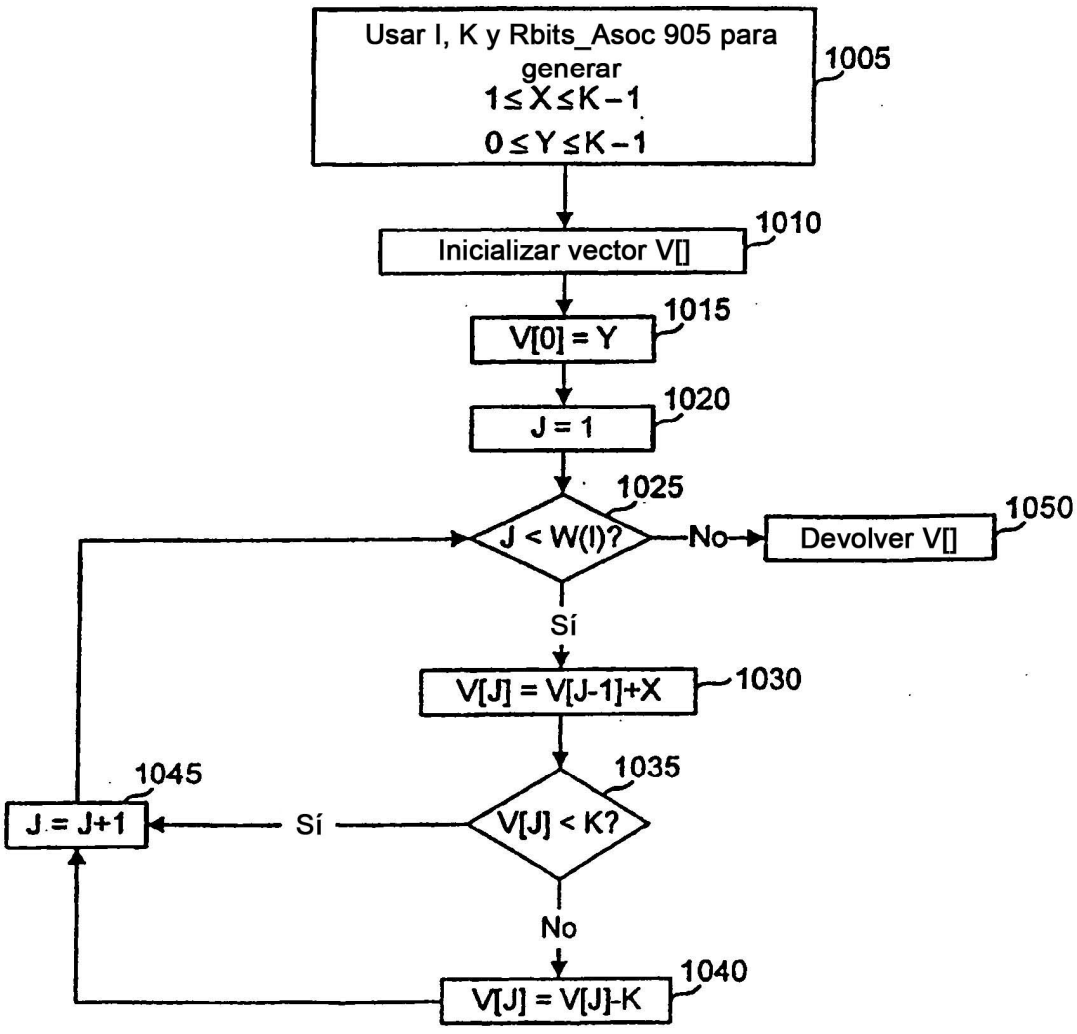

Figura 10

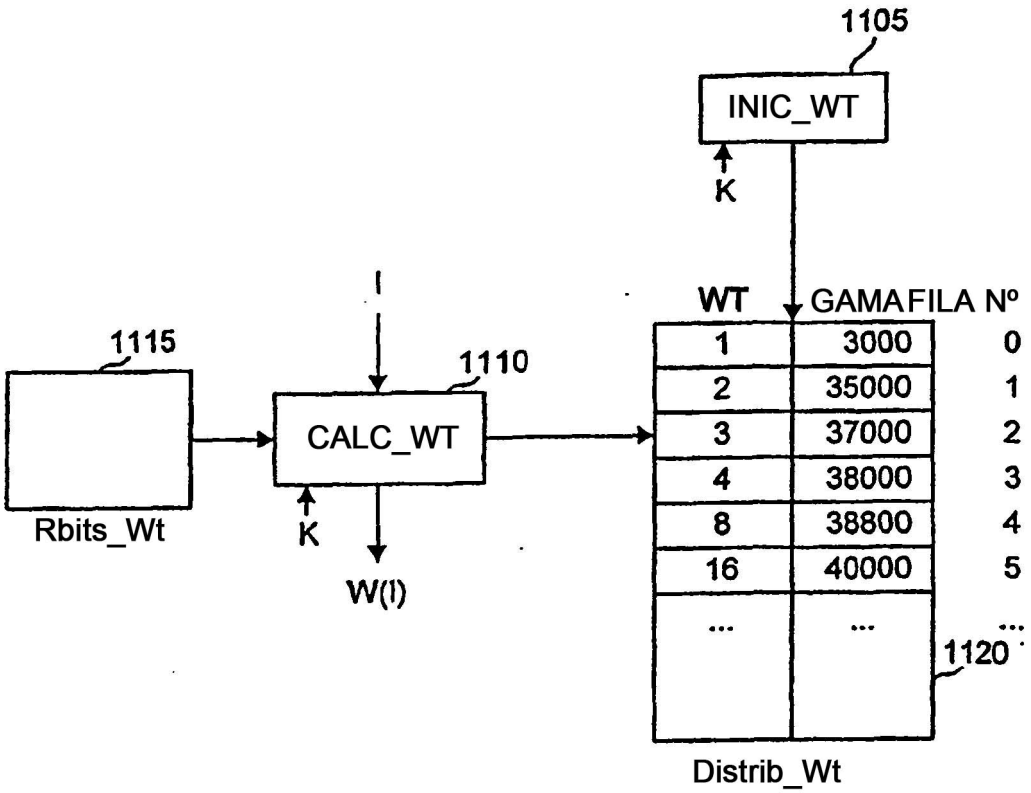

Figura 11

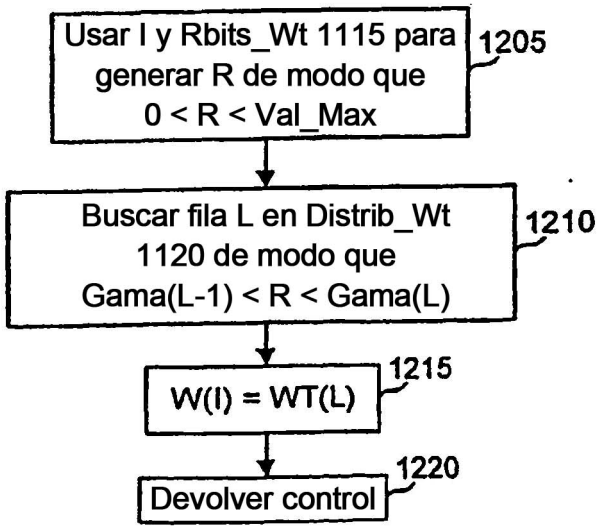

Figura 12

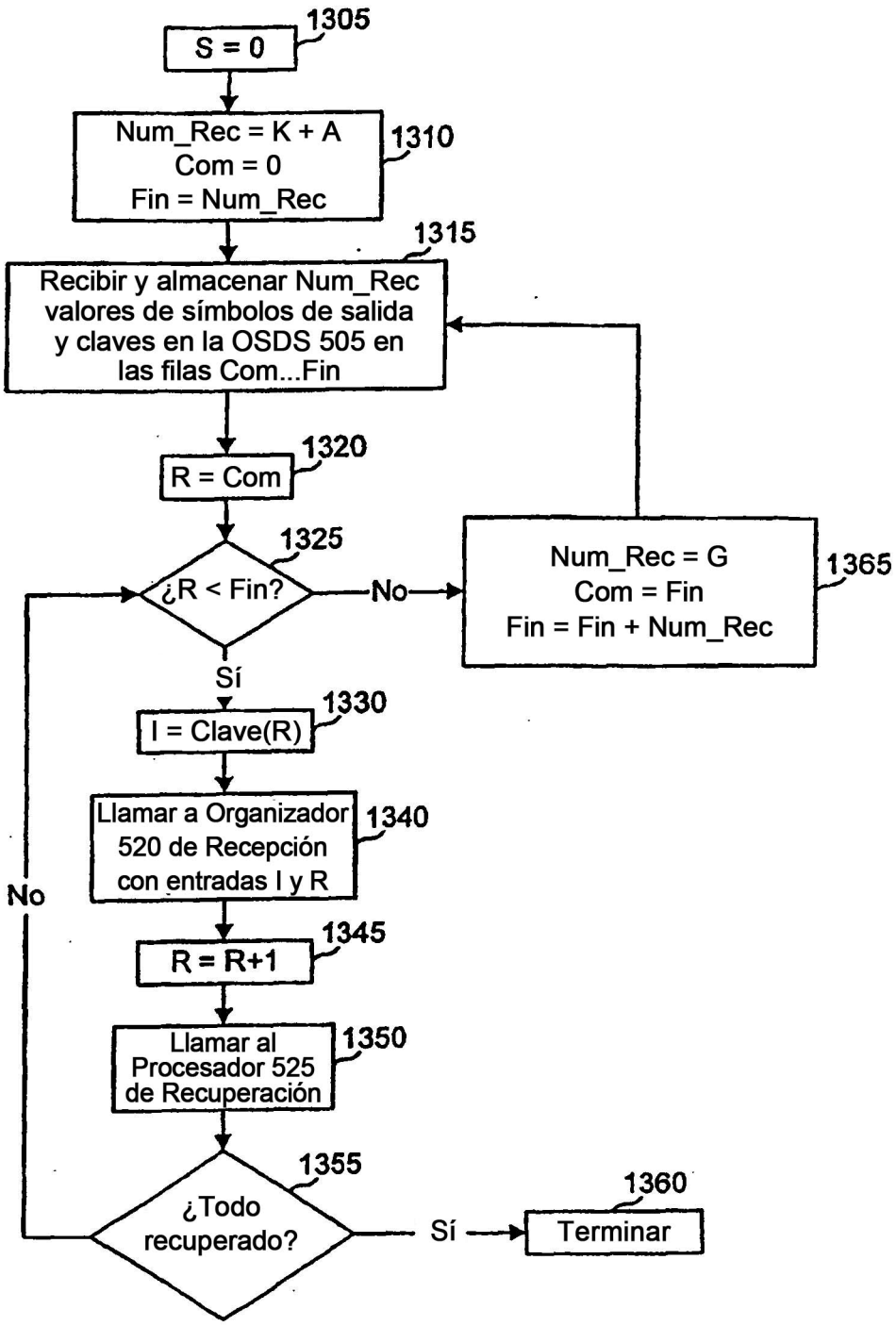
1305
S = 0
Num_Rec = K + A
Com = 0
Fin = Num_Rec
1310
1315
Recibir y almacenar Num_Rec valores de símbolos de salida y claves en la OSDS 505 en las filas Com...Fin
1320
R = Com
1325
¿R < Fin?
No
Num_Rec = G
Com = Fin
Fin = Fin + Num_Rec
1365
Sí
1330
I = Clave(R)
Llamar a Organizador 520 de Recepción con entradas I y R
1340
No
1345
R = R+1
Llamar al Procesador 525 de Recuperación
1350
1355
¿Todo recuperado?
Sí
1360
Terminar

Figura 13

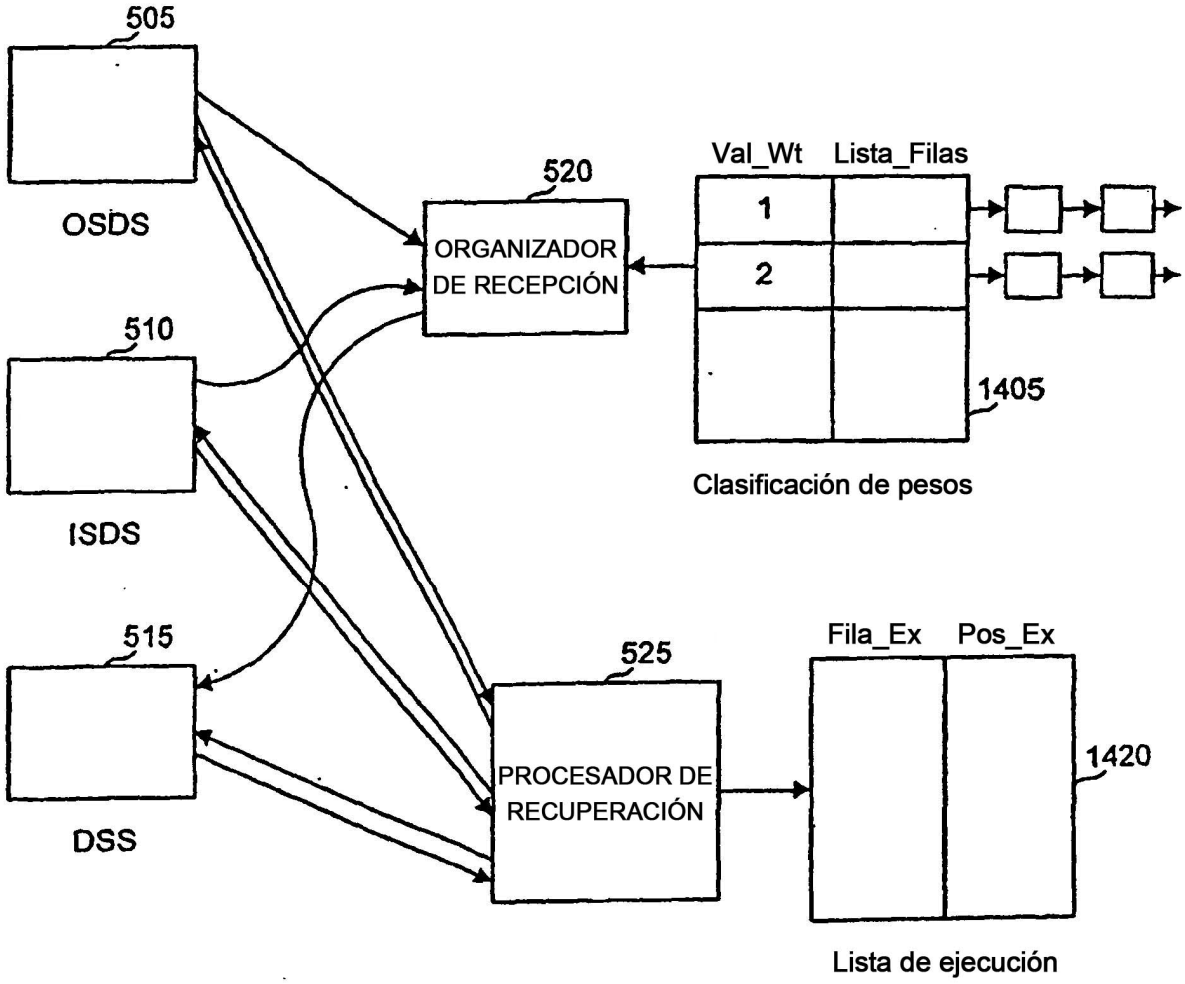
505
OSDS
510
ISDS
515
DSS
520
ORGANIZADOR DE RECEPCIÓN
Val_Wt
Lista_Filas
1
2
1405
Clasificación de pesos
525
PROCESADOR DE RECUPERACIÓN
Fila_Ex
Pos_Ex
1420
Lista de ejecución

Figura 14

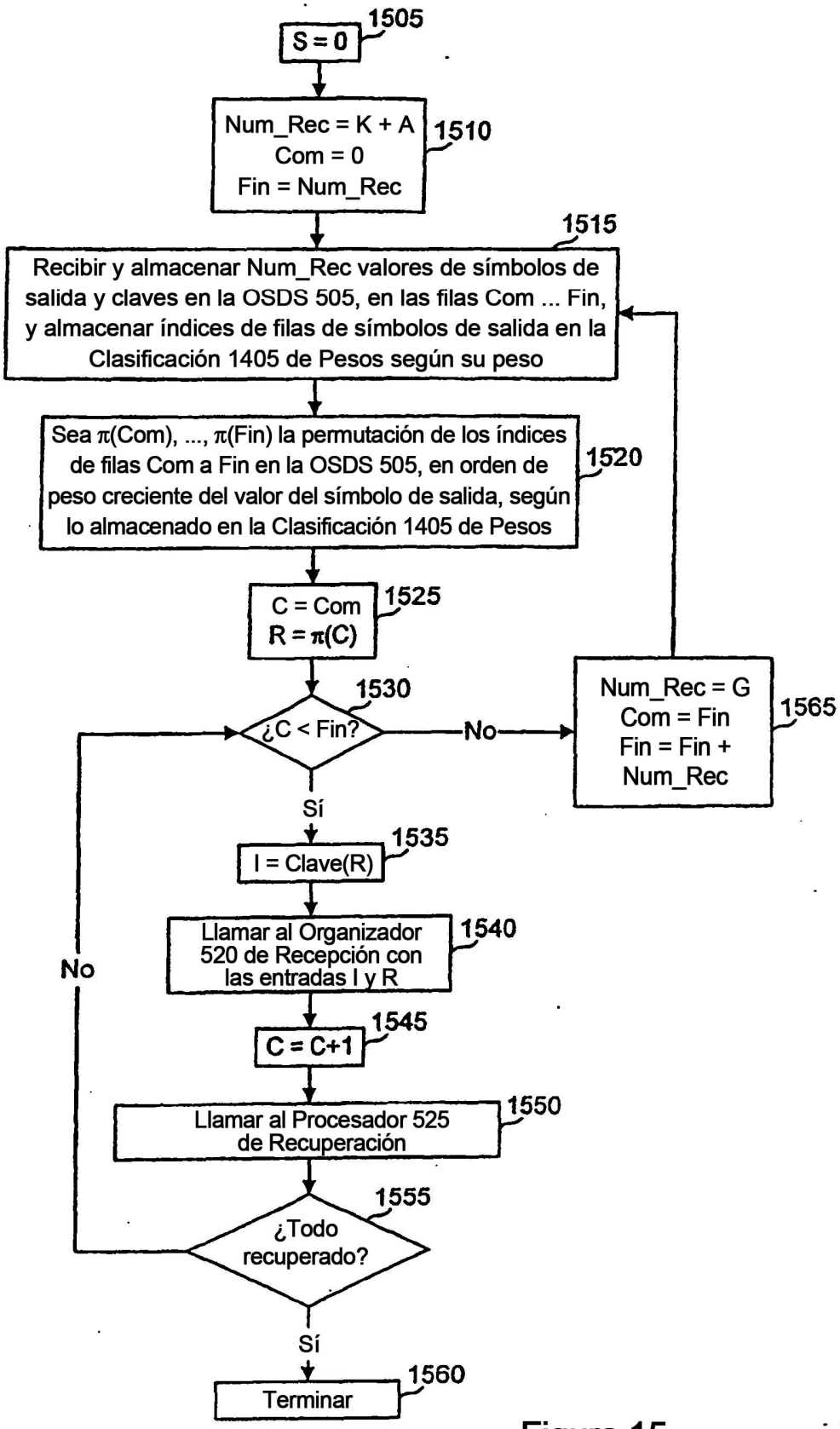
S = 0
1505
Num_Rec = K + A
Com = 0
Fin = Num_Rec
1510
1515
Recibir y almacenar Num_Rec valores de símbolos de salida y claves en la OSDS 505, en las filas Com ... Fin, y almacenar índices de filas de símbolos de salida en la Clasificación 1405 de Pesos según su peso
Sea π(Com), ..., π(Fin) la permutación de los índices de filas Com a Fin en la OSDS 505, en orden de peso creciente del valor del símbolo de salida, según lo almacenado en la Clasificación 1405 de Pesos
1520
C = Com
R = π(C)
1525
1530
¿C < Fin?
No
Num_Rec = G
Com = Fin
Fin = Fin + Num_Rec
1565
Sí
I = Clave(R)
1535
Llamar al Organizador 520 de Recepción con las entradas I y R
1540
No
C = C+1
1545
Llamar al Procesador 525 de Recuperación
1550
1555
¿Todo recuperado?
Sí
Terminar
1560

Figura 15

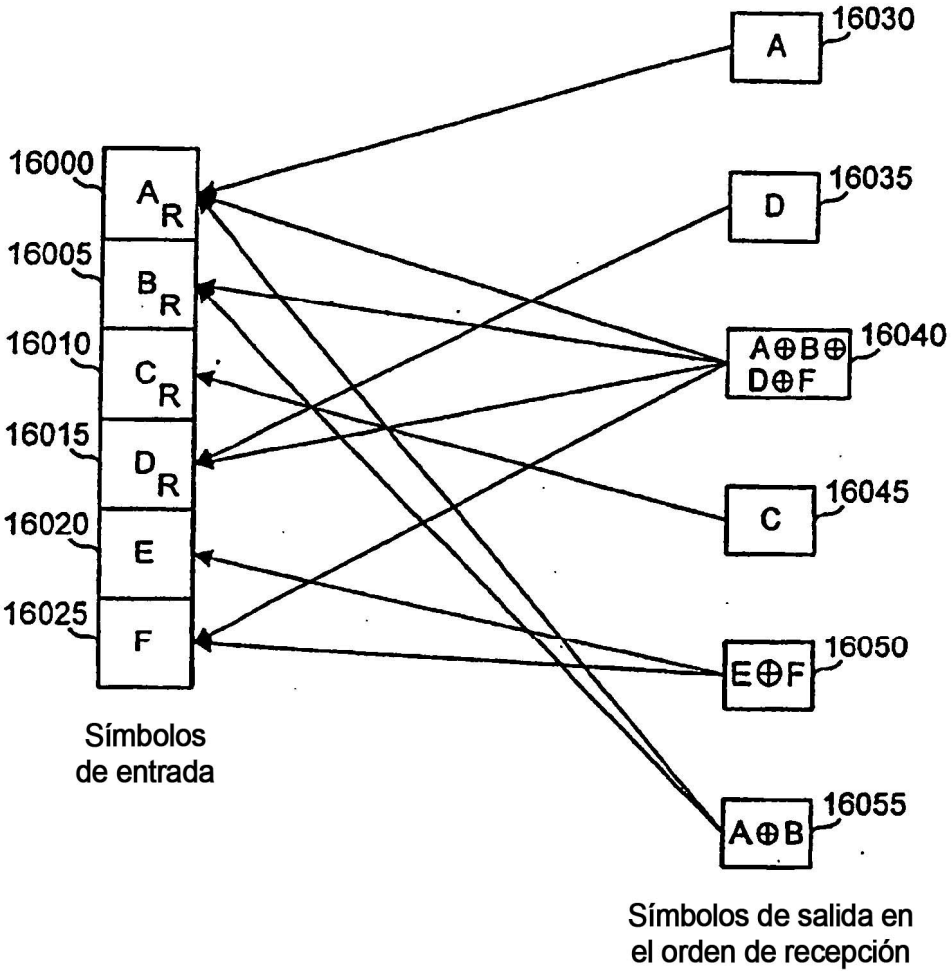
16000
A R
16005
B R
16010
C R
16015
D R
16020
E
16025
F
Símbolos de entrada
16030
A
16035
D
16040
A⊕B⊕ D⊕F
16045
C
16050
E⊕F
16055
A⊕B
Símbolos de salida en el orden de recepción

Figura 16

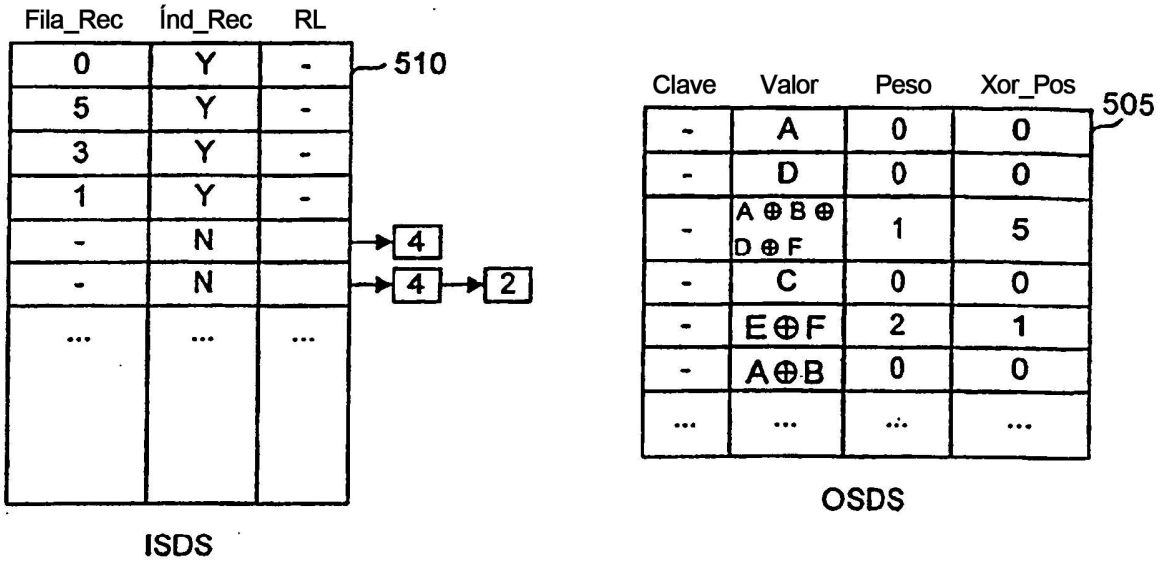

Figura 17

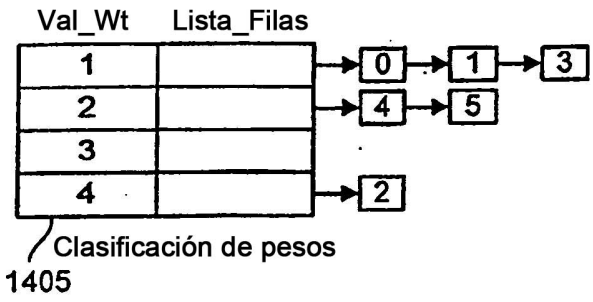

Figura 18

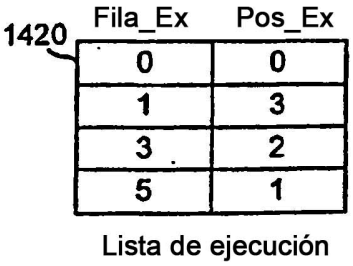

Figura 19

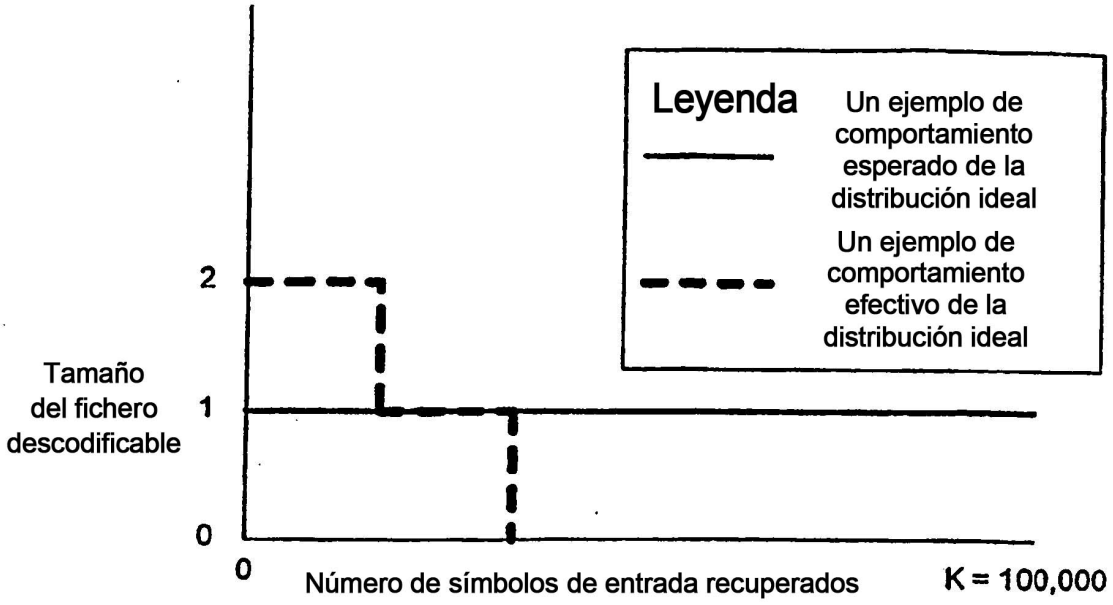

Figura 20

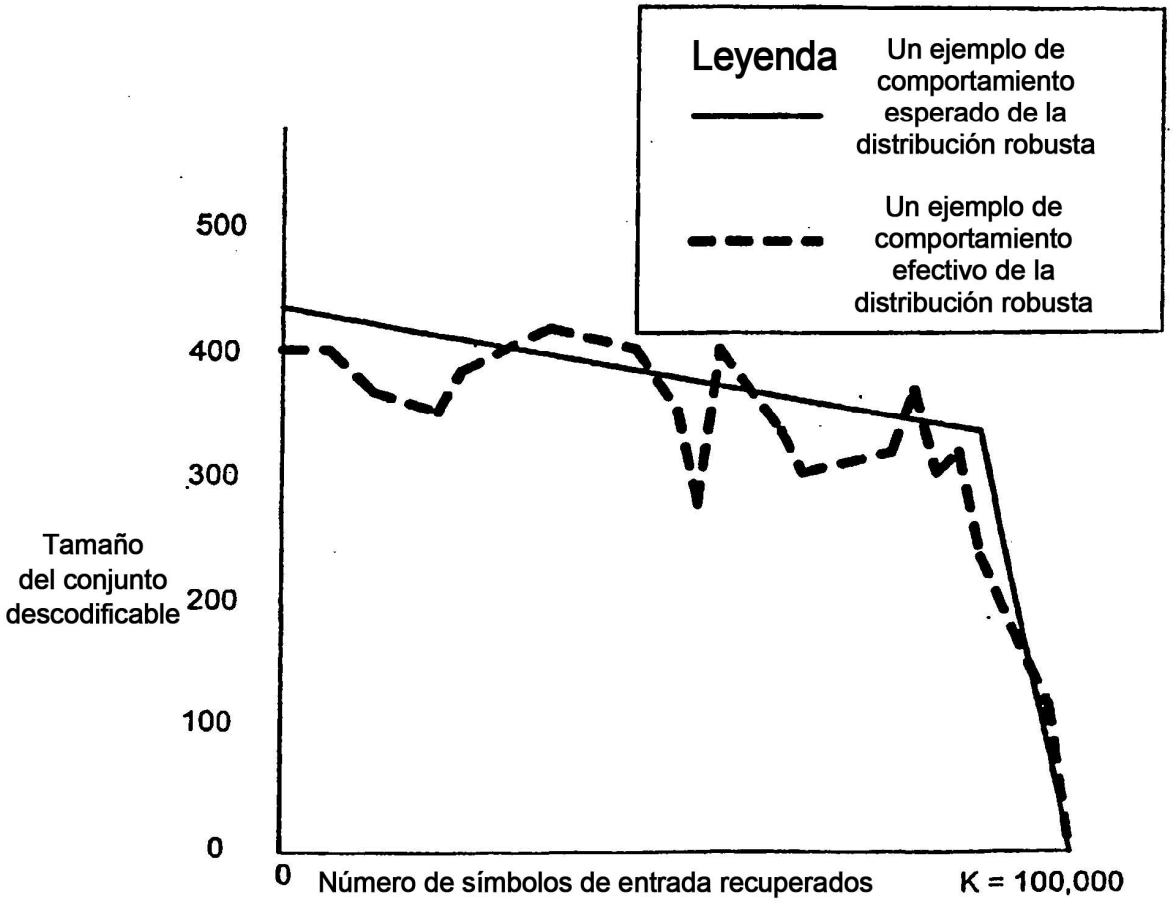

Figura 21

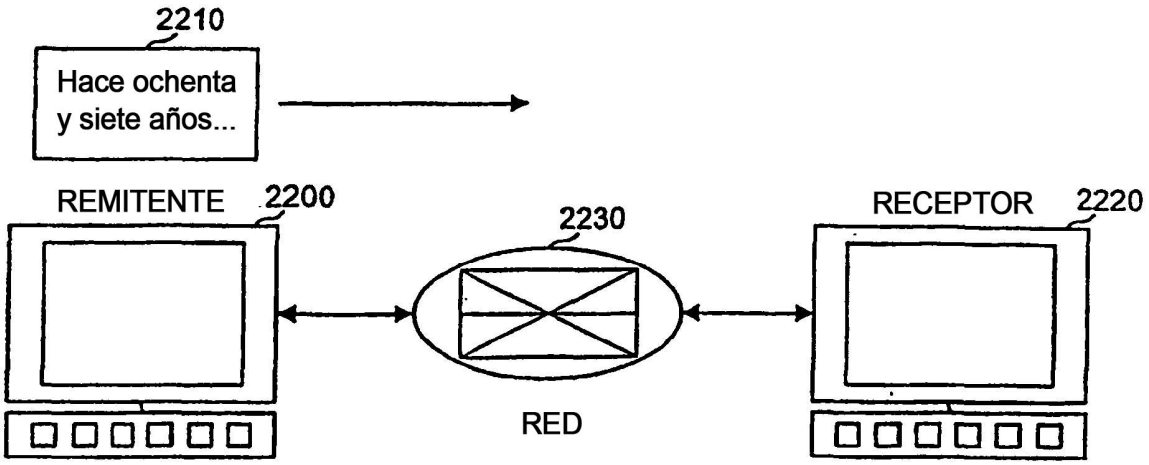

Figura 22

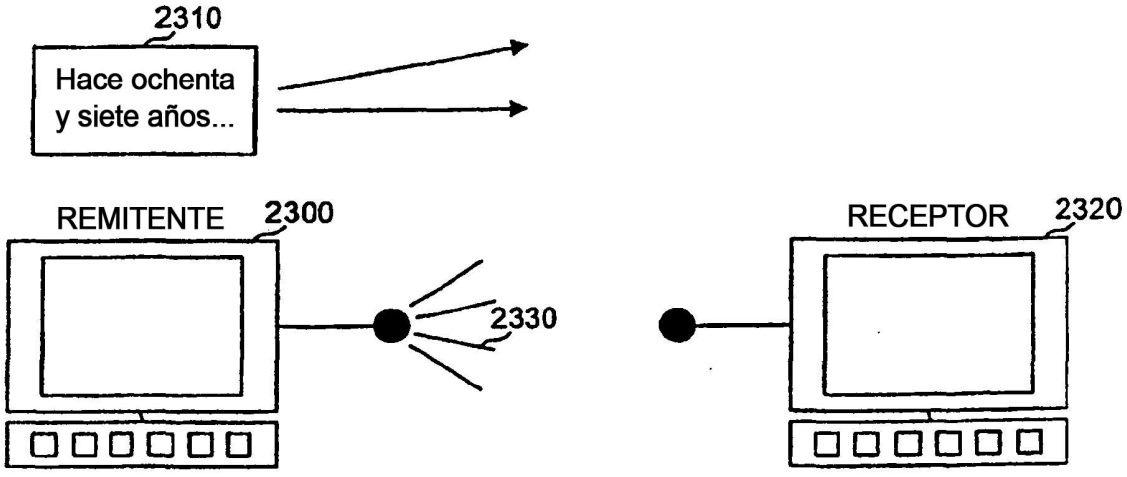

Figura 23

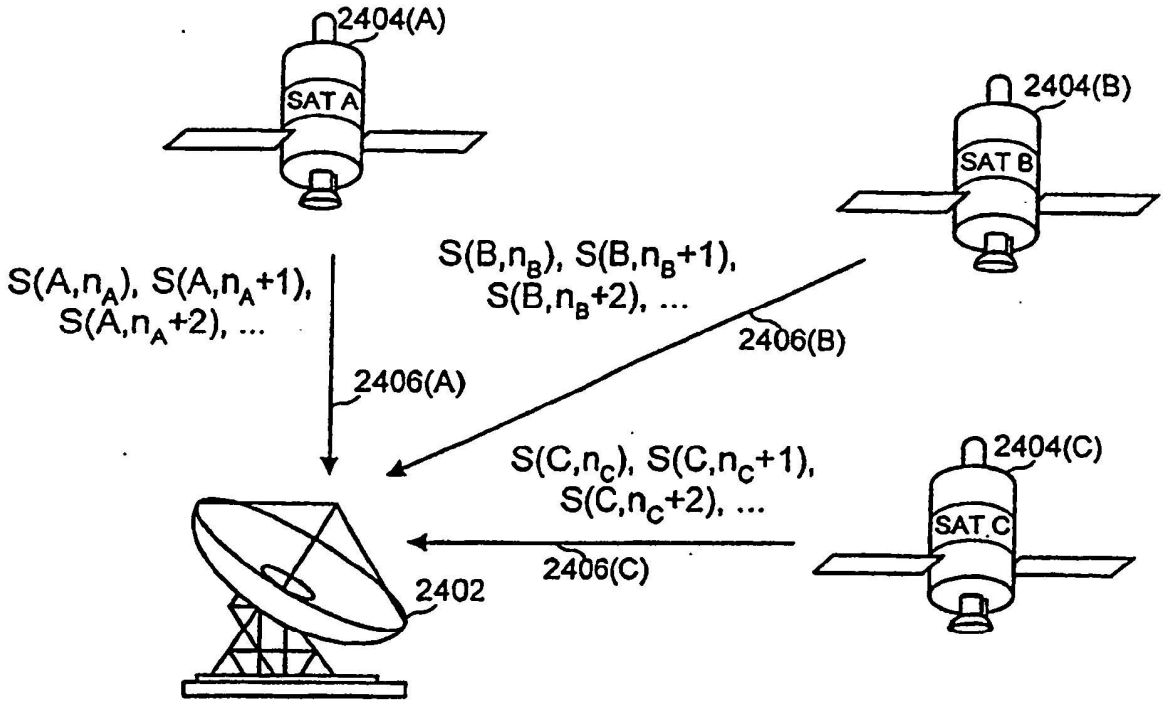

Figura 24

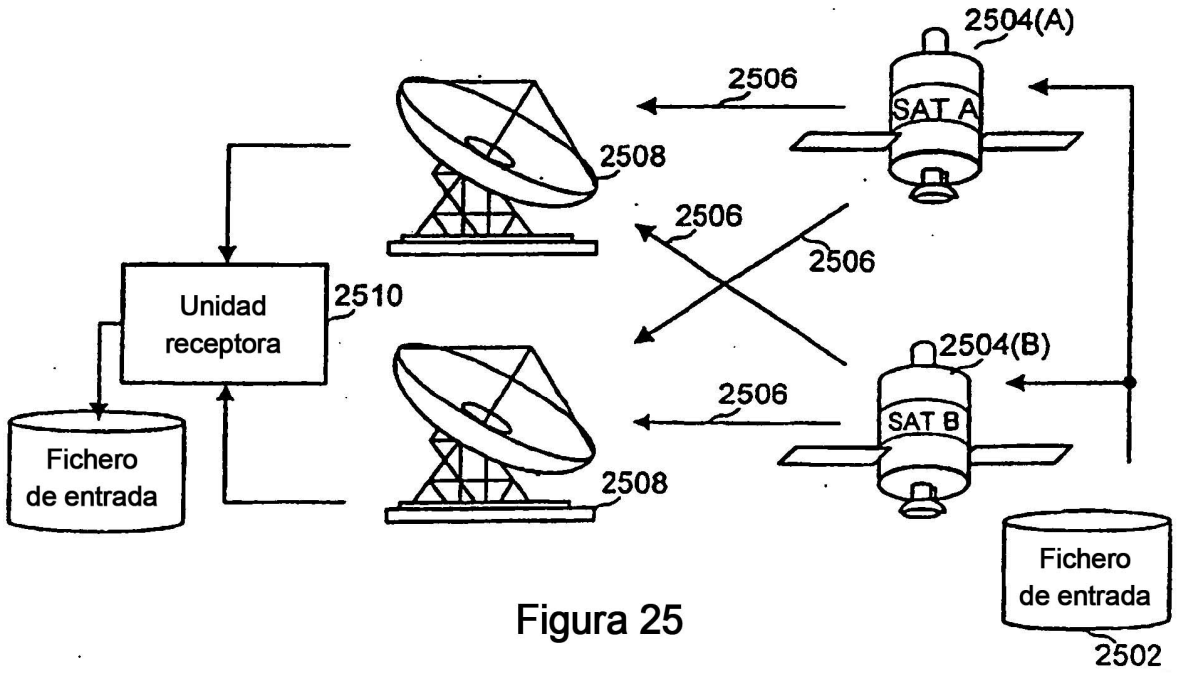

Figura 25