DETECCIÓN Y CORRECCIÓN DE ERRORES DE TRANSMISIÓN

Las redes de computadores deben ser capaces de transmitir datos de un dispositivo a otro con cierto nivel de precisión. Para muchas aplicaciones, el sistema debe garantizar que los datos recibidos son iguales a los trasmitidos. Sin embargo, siempre que una señal electromagnética fluye de un punto a otro, está sujeta a interferencias impredecibles debido al calor, el magnetismo y diversas formas de electricidad. Esta interferencia puede cambiar la forma o la temporización de la señal. Si la señal transporta datos binarios codificados, tales cambios pueden alterar su significado.

Las aplicaciones requieren entonces un mecanismo que permita detectar y corregir los posibles errores ocurridos durante la transmisión. Algunas aplicaciones tienen cierta tolerancia de errores (ej. transmisión de audio/video), mientras que para otras aplicaciones se espera un alto nivel de precisión (ej. transmisión de archivos).

En este documento se discuten algunos conceptos relacionados con la detección y corrección de errores en la transmisión de datos, así como algunas técnicas que llevan a cabo estas tareas.

1 Tipos de Errores

Antes de estudiar los mecanismos que permiten la detección y/o corrección de errores, es importante entender cuáles son esos posibles errores.

1.1 Error de Bit

Este término significa que únicamente un bit de una unidad de datos determinada (byte, caracter, paquete, etc.) cambia de 0 a 1 o de 1 a 0 [1][2]. Para comprender el impacto de este cambio, podemos imaginar que cada grupo de 8 bits es un caracter ASCII con un 0 añadido a la izquierda. Un error de bit podría alterar completamente el caracter ASCII enviado (ej. 'A': ASCII 65) y en el receptor se obtendría un caracter completamente diferente (ej. 'I': ASCII 73).

Los errores en un único bit son el tipo de error menos probable en la transmisión de datos en serie. Imagine que un emisor envía datos a 1Mbps. Esto nos dice que cada bit dura únicamente 1/1000000 seg. Para que ocurra un error de bit, el ruido debe tener una duración de sólo 1µseg, lo que es muy raro. Sin embargo, puede ocurrir un error de bit si se están enviando los datos usando transmisión paralela. Por ejemplo, si se usan 8 cables para enviar los 8 bits de un byte al mismo tiempo, y uno de los cables es ruidoso, se puede corromper un bit de cada byte.

1.2 Error de Ráfaga

Significa que dos o más bits de la unidad de datos han sido alterados. Es importante notar que los errores de ráfaga no implican que se afecten bits consecutivos. La longitud de la ráfaga se mide

desde el primer hasta el último bit incorrecto. Algunos bits intermedios pueden no estar afectados [1][2].

La Figura 3.1 muestra un ejemplo de error de ráfaga.

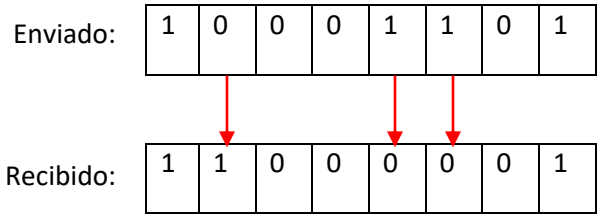

Figura 3.1. Error de ráfaga

En este caso, la longitud de la ráfaga sería 5, porque es la distancia en bits desde el primer bit erróneo hasta el último. Dentro de la ráfaga puede haber bits correctos y/o erróneos.

La presencia de errores de ráfaga es más probable en las transmisiones en serie. La duración del ruido es normalmente mayor que la duración del bit, lo que significa que cuando el ruido afecta los datos, afecta un conjunto de bits. El número de bits afectados dependerá de la tasa de datos y de la duración del ruido.

2 Redundancia

Una vez que se conocen los tipos de errores que pueden existir, es necesario identificarlos. En un entorno de comunicación de datos no se tendrá una copia de los datos originales que permita comparar los datos recibidos para detectar si hubo errores en la transmisión. En este caso, no habrá forma de detectar si ocurrió un error hasta que se haya decodificado la transmisión y se vea que no tienen sentido los datos recibidos. Si los computadores comprobaran errores de esta forma, sería un proceso muy lento y costoso. Es necesario un mecanismo que sea sencillo y completamente efectivo.

El concepto clave para detectar o corregir errores es la *redundancia*. Para esto es necesario enviar bits extra junto con los datos. Estos bits son añadidos por el emisor y eliminados por el receptor, permitiendo detectar y posiblemente corregir los bits afectados.

Un mecanismo de detección de errores que podría satisfacer los requisitos antes expuestos sería enviar dos veces cada unidad de datos. El dispositivo receptor podría entonces comparar ambas copias bit a bit. Cualquier discrepancia indicaría un error y se podría corregir mediante un mecanismo apropiado. Este sistema sería extremadamente lento. No solamente se doblaría el tiempo de transmisión, sino que además habría que añadir el tiempo necesario para comparar cada unidad bit a bit.

El concepto de incluir información extra en la transmisión con el único propósito de comparar es bueno. Pero en lugar de repetir todo el flujo de datos, se puede añadir un grupo más pequeño de bits al final de cada unidad. Esta técnica se denomina *redundancia* porque los bits extra son

redundantes a la información, descartándose tan pronto como se ha comprobado la exactitud de la transmisión [1][2].

3 Detección vs. Corrección

La corrección de errores es más difícil que la detección. En la *detección* sólo se quiere determinar si ha ocurrido un error, existiendo dos posibles respuestas: sí o no. La corrección como tal es sencilla, consiste tan solo en invertir los valores de los bits erróneos; sin embargo, es necesario previamente determinar la cantidad de bits erróneos, y aún más importante la ubicación de los mismos dentro de la unidad de datos [1][2].

La corrección de errores se puede conseguir de dos formas. En la primera, cuando se descubre un error, el receptor puede pedir al emisor que retransmita toda la unidad de datos (BEC, Backwards Error Correction). Con la segunda, el receptor puede usar un código corrector de errores, que corrija automáticamente determinados errores (FEC, Forward Error Correction).

En teoría, es posible corregir cualquier error automáticamente en un código binario. Sin embargo, los códigos correctores son más sofisticados que los códigos detectores y necesitan más bits de redundancia. El número de bits necesarios para corregir un error de ráfaga es tan alto que en la mayoría de los casos su uso no resulta eficiente [5].

4 FEC (Forward Error Correction) vs. Retransmisión

Como se mencionó previamente, existen dos mecanismos para la corrección de errores:

1. FEC: Forward Error Correction.
2. BEC: Backwards Error Correction.

FEC es el proceso en el que una vez detectado el error, el receptor trata de determinar el mensaje original, usando los bits de redundancia. Para esto es necesario incluir una mayor cantidad de bits de redundancia en la unidad de datos. BEC o *retransmisión* es la técnica en la que el receptor detecta la ocurrencia del error y solicita al emisor que reenvíe el mensaje. Se repite la retransmisión del mensaje hasta que el receptor compruebe que el mensaje ha llegado sin error (es posible que un error no sea detectado y el mensaje sea interpretado como correcto) [1][3].

Cada una de estas técnicas ocupa su nicho diferente [5]. En enlaces altamente confiables es más económico usar técnicas BEC, retransmitiendo los mensajes defectuosos que surjan eventualmente, sin necesidad de agregar una gran cantidad de bits de redundancia, lo que acarrearía una disminución de las prestaciones. Sin embargo, en enlaces poco confiables como los inalámbricos, puede resultar beneficioso agregar la redundancia suficiente a cada mensaje para que el receptor pueda reconstruir el mensaje original. Existen dos razones primordiales que sustentan el uso de las técnicas FEC:

1. La tasa de errores por bit en un enlace poco confiable puede ser muy grande, lo que resultará en un gran número de retransmisiones.

2. En algunos casos, el tiempo de propagación es muy elevado en comparación con el tiempo de transmisión. Por este motivo la retransmisión del mensaje resultaría muy costosa [3].

5 Códigos de bloque

Para entender la manera en que pueden manejarse los errores, es necesario estudiar de cerca cómo se codifican los datos. Por lo general, una unidad de datos (generalmente llamada en este ambiente *trama*) consiste de *m* bits de datos y *r* bits redundantes usados para la verificación, siendo la longitud total de una trama *n* (*n* = *m* + *r*). A la unidad de *n* bits que contiene datos y bits de redundancia se le conoce como *palabra codificada*. La cantidad de bits de redundancia y la robustez del proceso son factores importantes del esquema de codificación [1][2].

5.1 Distancia Hamming

Para empezar se define un concepto de utilidad. Se define la *distancia Hamming* $d(v_1, v_2)$ entre dos palabras codificadas de *n* bits v_1 y v_2, como el número de bits en el que v_1 y v_2 difieren [2]. Por ejemplo:

$v_1 = 10001001$; $v_2 = 10110001$ | entonces, $d(v_1, v_2) = 3$

5.1.1 Distancia Hamming mínima

Se llama *distancia Hamming mínima* a la distancia Hamming más pequeña entre todos los posibles pares de palabras codificadas de un esquema de codificación. Se usa el término d_{min} para definir la distancia Hamming mínima en un esquema de codificación. Para hallar este valor, se deben encontrar las distancias Hamming entre todas las palabras codificadas del esquema, y se selecciona la más pequeña [2].

5.2 Detección y corrección de errores mediante códigos de bloque

Las *palabras de datos* de longitud *m* bits no se transmiten directamente, sino que son previamente transformadas en *palabras codificadas* de *n* bits. Con *m* bits se pueden crear hasta 2^m palabras de datos, y con *n* bits se pueden crear hasta 2^n palabras codificadas. Como n>m, el número de palabras codificadas es mayor al número de palabras de datos. El proceso de codificación en bloques es uno-a-uno: la misma palabra de datos es transformada siempre en la misma palabra codificada. Las palabras codificadas obtenidas a partir de una palabra de datos son llamadas *válidas*. Esto significa que se tendrán $2^n - 2^m$ palabras codificadas que no serán utilizadas. Estas palabras codificadas son llamadas *inválidas* [1].

5.2.1 Detección de errores con códigos de bloque

Ahora, ¿cómo puede usarse la codificación por bloques para detectar errores? Si se cumplen las siguientes dos condiciones, el receptor será capaz de detectar variaciones en la palabra codificada original:

1. El receptor tiene la lista de las palabras codificadas válidas.
2. La palabra codificada válida transmitida ha sido modificada a una inválida.

La Figura 3.2 muestra cómo se pueden detectar errores en la codificación por bloques.

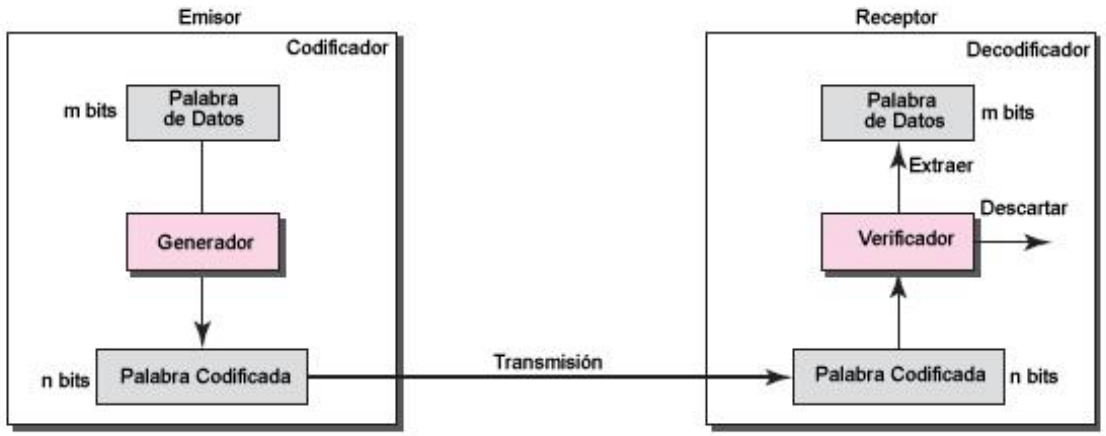

Figura 3.2. Proceso de detección de errores en la codificación por bloques

El emisor crea palabras codificadas a partir de palabras de datos usando un generador que aplica reglas y procedimientos de codificación específicos del esquema empleado. Cada palabra codificada que es enviada al receptor puede variar durante la transmisión. Si la palabra codificada recibida no es válida, es descartada. Sin embargo, si la palabra codificada es modificada como otra palabra codificada válida durante la transmisión, el error no será detectado.

5.2.2 Corrección de errores con códigos de bloque

En el caso discutido previamente (detección de errores), el receptor sólo necesita saber que la palabra codificada es inválida para detectar un error. En la corrección de errores, el receptor deberá descubrir la palabra codificada originalmente enviada. La idea principal es la misma que la empleada en la detección de errores, pero el verificador es mucho más complejo. Se aprecia el funcionamiento del proceso de corrección en la Figura 3.3 [1].

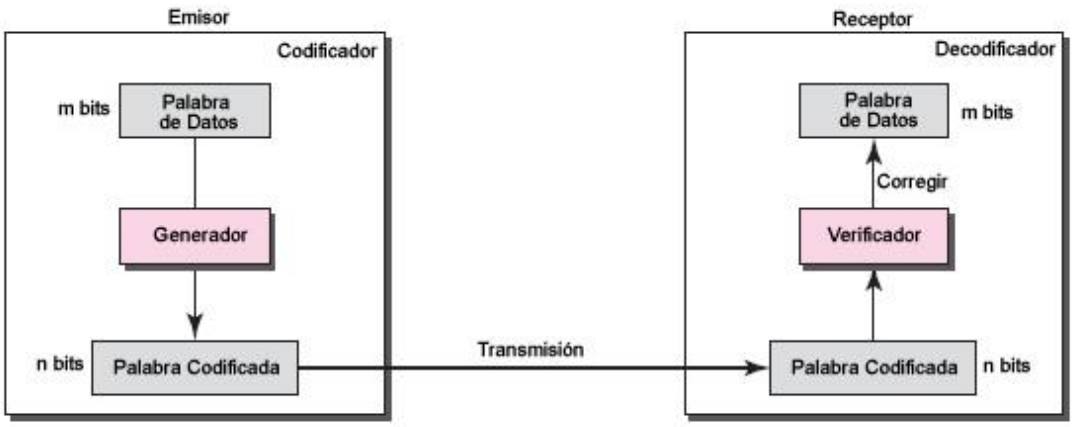

Figura 3.3. Proceso de corrección de errores en la codificación por bloques

Una vez que se recibe una palabra inválida, el receptor calcula la distancia Hamming entre la palabra recibida y las palabras válidas. La menor de las distancias calculadas indica cual es la palabra codificada válida que el emisor originalmente transmitió. Si dos o más palabras válidas generan el mismo valor, que resulta ser el mínimo, entonces el error no puede ser corregido y la palabra recibida se descarta.

En la mayoría de las aplicaciones de transmisión 2^m palabras son válidas, pero como se ha visto, debido a la manera en que se codifican no se usan las 2^n palabras codificadas posibles. Es viable entonces hacer una lista de las palabras codificadas válidas y encontrar las dos cuya distancia Hamming sea mínima. Esta será la distancia Hamming de todo el código [1][5].

Las propiedades de detección y corrección de errores de un código dependen de su distancia Hamming. Si dos palabras codificadas están separadas una distancia Hamming *d*, se requerirán *d* errores de un bit para convertir una en otra.

Para detectar *d* errores se necesita un código de distancia $d + 1$, pues con tal código no habrá manera de que *d* errores de bit puedan cambiar una palabra codificada válida a otra. Cuando el receptor encuentra una palabra codificada no válida, sabe que ha ocurrido un error de transmisión. De manera similar, para corregir *d* errores se necesita un código de distancia $2d + 1$, pues así las palabras codificadas válidas estarán tan separadas que, aún con *d* cambios, la palabra codificada original sigue estando más cercana que cualquier otra palabra codificada, por lo que puede determinarse de manera única [5].

El Ejemplo 3.1 muestra dos sencillos códigos de bloque para la detección y corrección de errores de un bit:

Ejemplo 3.1:

Código para detección de errores		Código para la corrección de errores	
Palabra de Datos	Palabra Codificada	Palabra de Datos	Palabra Codificada
00	000	00	00000
01	011	01	01011
10	101	10	10101
11	110	11	11110

El ejemplo muestra la notable diferencia en la cantidad de bits de redundancia necesarios entre una técnica de detección en contraste con los necesarios en una técnica de corrección de errores.

6 Códigos cíclicos

Los códigos cíclicos son códigos de bloque que cumplen varias propiedades:

- Al aplicar la operación XOR sobre dos palabras codificadas válidas, se genera otra palabra codificada válida.
- Si una palabra codificada válida es rotada en forma cíclica, el resultado es otra palabra codificada válida.

Por ejemplo, si se tiene la palabra codificada 1001101001, y se rota hacia la izquierda de forma cíclica, entonces 0011010011 debe ser una palabra codificada válida [1][3].

6.1 CRC (Cyclic Redundancy Check)

Es una categoría de códigos cíclicos ampliamente usada en redes LAN y WAN. En la codificación, la palabra de datos de *m* bits se combina con *r* bits en cero (0), de tal forma que $n = m + r$. Los *n* bits resultantes se pasan al *generador CRC*. El generador usa un divisor de $r + 1$ bits, que debe ser predefinido y compartido entre el emisor y el receptor. El generador luego divide la palabra de *n* bits por el divisor usando aritmética módulo 2. El cociente de la división se descarta, y el resto se concatena con la palabra de datos generando así la palabra codificada [1].

En la decodificación (receptor) se recibe la palabra posiblemente alterada por los errores de transmisión. Se ingresa la palabra de *n* bits al verificador y se divide nuevamente por el mismo divisor usado previamente. El resto producido por la división es un *síndrome* de *r* bits que se entrega al analizador lógico. Este analizador cumple una simple función: si los bits del síndrome son todos ceros (0), los primeros *m* bits de la palabra codificada son aceptados como la palabra de datos original; si al menos un bit es distinto de cero se descarta la palabra porque se asume que ha ocurrido un error. El proceso se observa en la Figura 3.4.

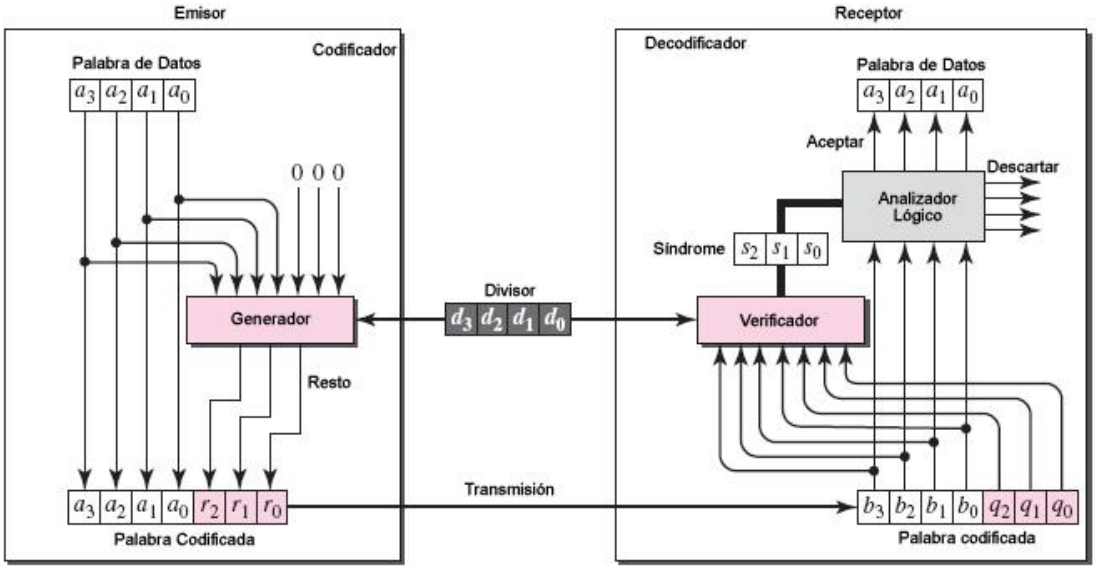

Figura 3.4. Codificador y Decodificador CRC

6.1.1 El Generador del CRC

Un generador de CRC usa división módulo 2. En el Ejemplo 3.2 se ilustra el proceso. En el primer paso, al divisor de cuatro bits se le aplica *OR Exclusivo* (**XOR**) con los cuatro primeros bits del dividendo. Esta operación no afecta al bit que está a continuación. En el ejemplo, el divisor 1101 ^ 1001 (los cuatro primeros bits del dividendo) arrojan el resultado 100 (el 0 inicial se descarta) [1].

A continuación se arrastra el siguiente bit sin usar del dividendo para hacer que el número de bits sea igual al del divisor. Por tanto, el paso siguiente (1101 ^ 1000) arroja 101, continuando el proceso de la misma forma.

Si el bit más a la izquierda del resto es cero (0), no se puede usar el divisor compartido, sino que se debe usar una cadena de ceros de la misma longitud. Este proceso se mantiene hasta que se hayan usado todos los bits en el dividendo.

Ejemplo 3.2:

Datos = 100100
Datos + ceros extra = 100100000
Divisor = 1101 (Se comparte entre emisor y receptor)

```
1 0 0 1 0 0 0 0 0
1 1 0 1
- 1 0 0 0
  1 1 0 1
  - 1 0 1 0
    1 1 0 1
    - 1 1 1 0
      1 1 0 1
      - 0 1 1 0
        0 0 0 0
        - 1 1 0 0
          1 1 0 1
          - 0 0 1
```

En este caso, se divide por una secuencia de ceros porque el resultado de la división comienza con un bit cero.

Resto = 001

Unidad de datos a enviar = 100100001

6.1.2 El Comprobador CRC

Este funciona igual que el generador. Después de recibir los datos junto con el CRC, hace la división módulo 2 y se entrega el resto al analizador lógico. Si todo el resto son ceros, el resto CRC se descarta y se aceptan los datos; en cualquier otro caso, el flujo de bits recibido se descarta y se solicita la retransmisión de los datos [1].

6.1.3 Implementación del CRC en Hardware

El codificador y decodificador CRC pueden ser implementados muy fácil y económicamente vía hardware. Además, una implementación por hardware incrementaría la velocidad del cálculo de la división. A continuación se describe el proceso de implementación, mediante dispositivos electrónicos, de los elementos involucrados en el cálculo del CRC [1][2].

Primero consideremos el divisor. Es importante notar algunos factores:

1. Al divisor se le aplica repetidamente la operación XOR con una porción del dividendo.
2. El divisor tiene $r + 1$ bits (o lo que es lo mismo, $n - m + 1$ bits) que están predefinidos, o son ceros. Estos bits no varían sin importar cuál sea la palabra de datos.
3. Sólo son necesarios $n - m$ bits del divisor para la operación XOR. El bit más significativo (más a la izquierda) no es necesario ya que el resultado siempre será cero, sin importar cuál sea su valor. La razón es que la entrada de la operación del XOR serán dos bits iguales (1 o 0).

Basándose en estos puntos, es posible diseñar un divisor cableado fijo que sea usado para un código cíclico si se conoce el patrón del divisor. Se construye un circuito de desplazamiento basándose en el divisor acordado, según en los siguientes puntos [1][2]:

1. Se tiene un flip-flop para cada bit del divisor, excepto el más significativo (*r* – 1 flip-flops).
2. Se coloca una compuerta XOR en la entrada de cada flip-flop cuya posición correspondiente en el divisor sea un bit 1.
3. Se hace una retroalimentación de la salida del flip-flop correspondiente al segundo bit más significativo (el primero no se representa con flip-flop) hacia todas las compuertas XOR presentes en el circuito.

La Figura 3.5a muestra el diseño general del circuito para codificador; mientras que la 3.5b muestra el decodificador.

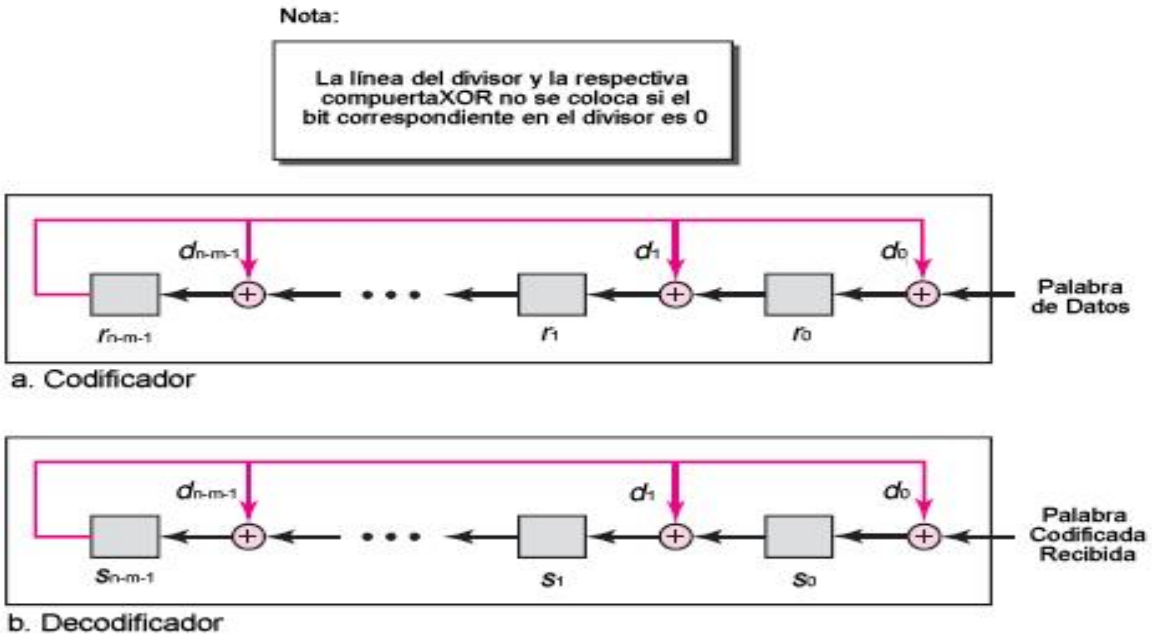

Figura 3.5 Diseño general del codificador y decodificador CRC por hardware

El proceso para obtener el resto mediante el uso del circuito se divide en los siguientes pasos [1][2]:

1. Se asume que el contenido original de los registros es cero.
2. En cada señal de reloj (llegada de un bit de la palabra de datos aumentada con ceros), se repiten las siguientes acciones:
    a. Se aplica XOR en cada compuerta presente con las entradas respectivas.
    b. Se desplazan las entradas de registro en registro hacia la izquierda.
3. Una vez que se han tratado todos los bits de la palabra de datos aumentada, el contenido final de los registros será el resto CRC que debe concatenarse con la palabra de datos.

6.2 Ventajas de los códigos cíclicos

Los códigos cíclicos son ampliamente usados por su sencillez y eficacia. Pueden fácilmente implementarse tanto en hardware como en software. Son muy rápidos cuando se implementan por hardware, con elementos de bajo costo (registros y compuertas).

7. Suma de Verificación: Checksum

El último método de detección de errores que será discutido es *checksum*. El checksum es usado en Internet por diversos protocolos de capas superiores [1]. Como la mayoría de las técnicas de control de errores, el checksum se basa en la redundancia; se agregan bits de redundancia acompañando a los bits de datos. El valor de los bits de redundancia es calculado a partir de los bits de datos. El proceso se describe a continuación.

7.1 Cálculo del checksum

La idea principal consiste en subdividir la unidad de datos en segmentos de igual longitud, *k* bits (16 habitualmente). Estos segmentos se suman y el resultado de la suma será el valor de los bits de redundancia. En el receptor, se calcula nuevamente la suma y se compara con el valor de los bits de redundancia; si coinciden, se asume que no hay error. Para facilitar el proceso en el extremo receptor, se envía como datos de redundancia el valor negativo (complemento) de la suma. De esta forma, el receptor debe sumar todos los datos más los de redundancia, si el resultado es cero, el receptor recibe los datos como correctos [1].

7.1.1 El generador del checksum

Existe una limitante en el esquema planteado, y es la longitud (en bits) de la representación del resultado, que dependerá de la magnitud del mismo. Una solución sencilla para este problema es usar aritmética *complemento a 1*. Esta aritmética permite representar números sin signo entre 0 y 2^n-1 usando sólo n bits [1]. Si el número excede los n bits, los bits más significativos sobrantes se suman a los menos significativos. A continuación, se complementa ese total (en la aritmética complemento a 1, los números negativos se representan invirtiendo los valores de sus bits, es decir, los 0s se convierten en 1s y viceversa) y se añade al final de la unidad de datos originales como bits de redundancia. Finalmente, la unidad de datos extendida se transmite a través de la red. Si la suma del segmento de datos es T, la suma de comprobación sería –T.

7.1.2 El comprobador del checksum

El receptor subdivide las unidades de datos (en bloques de la misma longitud usada por el emisor) como se explicó anteriormente, suma todos estos segmentos y complementa el resultado. Si la unidad de datos extendida está intacta, el valor total que se obtiene al sumar los segmentos de datos y el campo de suma de comprobación debe ser 0. Si el resultado no es 0, el paquete contiene un error y el receptor lo rechaza.

En el Ejemplo 3.3 se muestra un cálculo sencillo de checksum.

Ejemplo 3.3: Se asume que los datos a ser verificados son los siguientes: 0x0123456789ABCDEF (mostrados en hexadecimal por comodidad). Se dividen en bloques de 16 bits y se suman de la siguiente forma:

0123
4567
89AB
CDEF
19E24 → Excede 16 bits, se aplica aritmética complemento a 1

9E24
0001
9E25 → Resultado de la suma. Se complementa, y se obtiene: **61DA**

7.2 Checksum de IP

El checksum que se coloca en la cabecera IP se calcula basándose sólo en los datos de la cabecera, sin incluir el *payload*. Para calcular el checksum de un datagrama saliente, el valor del campo checksum se fija en 0. Luego se considera la cabecera como una secuencia de palabras de 16 bits y se suman todos los bloques (palabras), al resultado de la suma se le calcula el complemento a 1. El complemento a 1 en 16 bits del resultado de la suma se almacena en el campo de checksum [4].

Cuando un datagrama IP es recibido, se calcula el complemento a 1 de la suma de los bloques de 16 bits de la cabecera. Como el checksum calculado por el receptor contiene el valor almacenado por el emisor, el resultado debe ser 16 bits en 1 si no ocurrió ningún cambio en la cabecera (al aplicar el complemento, el resultado es 0). Si el resultado no está compuesto por 16 bits en 1 se dice que ocurrió un error de checksum, por lo que IP descarta el datagrama en cuestión. No se genera ningún mensaje de error; es deber de las capas superiores detectar los datos faltantes y recuperar el error.

7.3 Checksum de UDP y TCP

TCP y UDP incluyen en su cálculo de checksum varios campos de la cabecera IP, así como sus propias cabeceras y los datos. A continuación se discuten ambos casos.

7.3.1 Checksum de UDP

El checksum de UDP cubre la cabecera UDP y el payload [4]. En UDP el checksum es opcional (se envían todos los bits en 0 para indicar que no fue calculado). A pesar de que el cálculo básico del checksum es similar al explicado previamente (checksum de IP), existen algunas diferencias. Primero, la longitud del datagrama UDP puede ser impar, y el algoritmo está diseñado para sumar palabras de 16 bits. Una solución sencilla es agregar un relleno compuesto de 0s al final para el cálculo del checksum. Este relleno no sería transmitido y se descarta una vez calculada la suma.

También, UDP usa una pseudo-cabecera de 12 bytes sólo para el cálculo del checksum [4]. Esta pseudo-cabecera incluye algunos campos de la cabecera IP (ej.: source address, destination address, protocol). El propósito es facilitar una doble verificación desde UDP para garantizar que los datos han llegado al destino correcto. Si el receptor detecta un error, el datagrama es descartado y no se genera un mensaje de error.

7.3.1 Checksum de TCP

El checksum es obligatorio en TCP, y cubre todo el segmento: la cabecera y el payload. El checksum es calculado de forma similar a UDP, usando una pseudo-cabecera de 12 bytes [4].

El checksum de UDP y TCP es llamado *end-to-end*, porque es calculado y verificado por los nodos origen y destino de la comunicación.

Referencias

1. Forouzan, B. Data Communications and Networking. Mc Graw Hill. 4th ed. 2007.
2. Stallings, W. Comunicaciones y Redes de Computadores. Prentice Hall. 7ª ed. 2004.
3. Stallings, W. Wireless Communications and Networks. Prentice Hall. 2001.
4. Stevens, R. TCP/IP Illustrated, Volume 1, The Protocols. Addison Wesley. 1994.
5. Tanenbaum, A. Redes de Computadores. Prentice Hall. 4ª ed. 2003.