Capítulo 3 Capa de enlace de datos

Resumen

La capa de enlace de datos se encarga de enviar una secuencia de datos acotada (una PDU, *Protocol Data Unit*) entre dos dispositivos a través de un canal de comunicación o enlace. Los dispositivos pueden ser un equipo terminal y un nodo de conmutación o dos nodos de conmutación intermedios (también pueden ser dos equipos terminales, pero en este caso, no tenemos propiamente una red). A la secuencia de datos suele llamársele trama en esta capa.

Además de formar la trama, esta capa suele encargarse de verificar que llega al siguiente dispositivo sin errores, o por lo menos de detectar que ha ocurrido un error durante la transmisión. Los principios básicos utilizados para la detección de errores (y la eventual retransmisión en caso de que ocurran) se presentan en este capítulo. Estos conceptos se retoman y amplían en el capítulo correspondiente a la Capa de Transporte, donde se verifica la integridad de la información de extremo a extremo, es decir, entre equipos terminales.

3.1 Introducción

Como se ha comentado en capítulos anteriores, las señales que se propagan por un canal de comunicaciones están sujetas a una serie de afectaciones, como distorsión, atenuación e interferencia, por lo que siempre hay una probabilidad (a veces alta) de que lo que se recibe sea distinto a lo que se envía, es decir, que haya ocurrido un error.

La función principal de la capa de enlace de datos, es precisamente detectar la ocurrencia de esos errores y, en ocasiones, recuperarse de esos errores ya sea porque la información se retransmite o, como veremos en un momento, porque se envían datos adicionales (redundantes) que permiten la auto-corrección.

Esta detección de errores se da entre equipos o nodos conectados directamente. Estos pueden ser un equipo terminal (como una computadora, un servidor de archivos, una impresora) y un equipo intermedio (como un modem, un *Access Point*, un conmutador, un enrutador) o entre equipos intermedios.

Por razones históricas, los equipos terminales se conocen como DTE (*Data terminal equipment*) y son las fuentes o destinatarios de los datos a intercambiar. Entre ellos y la red, debe haber un dispositivo que adapta los datos a las características del servicio ofrecido por la red (imagine actualmente un proveedor de acceso a Internet). A ese dispositivo se le conoce como DCE (*Data circuit-terminating equipment o data communications equipment*). La figura 3.1 muestra dónde se ubican los DTE y DCE en un *circuito de datos*, la trayectoria entre dos DCE conectados a equipos terminales.

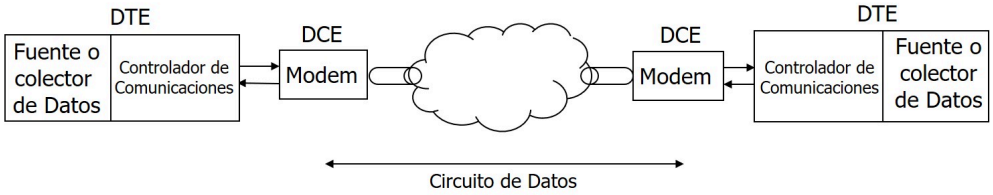

Figura 3.1: *Circuito de datos.*

En términos del modelo de OSI, la capa de enlace de datos ofrece sus servicios a la capa de red. Recibe de y le entrega a ésta una unidad de datos (un SDU) que, recordemos, en la capa de red se llama datagrama o paquete. Para ofrecer el servicio de detección (y posiblemente corrección) de errores, es que se requiere de un protocolo en la capa de enlace de datos, como se aprecia en la figura 3.2.

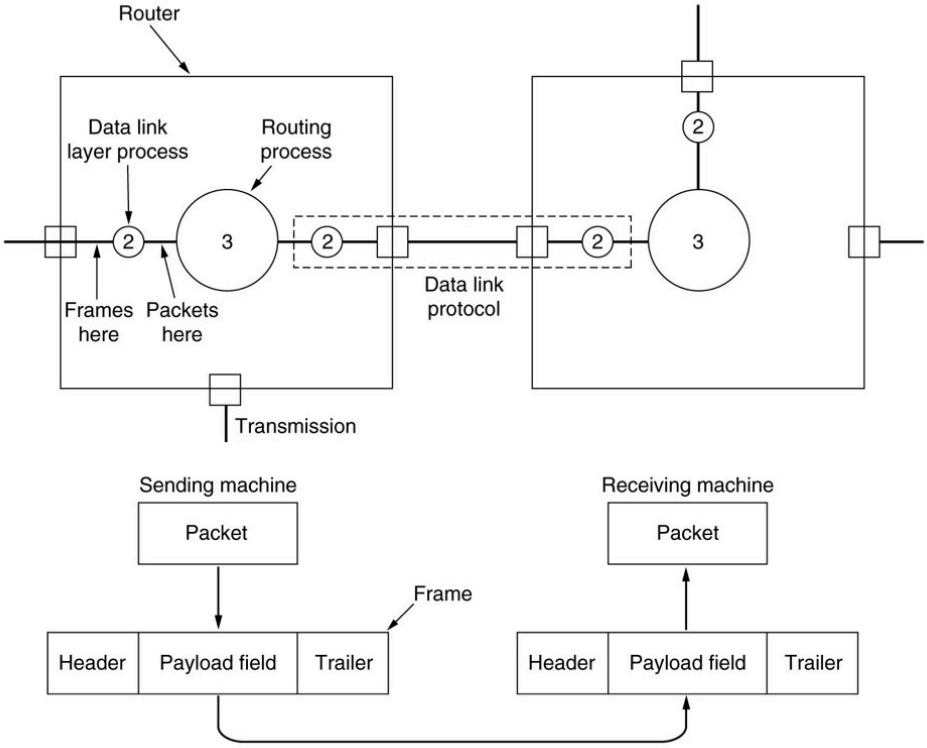

Figura 3.2: *La capa de enlace de datos presta sus servicios a la capa de red.*

En el emisor, esta capa recibe un datagrama de su capa superior y, en principio, debe entregar uno prácticamente idéntico a la capa superior del receptor. Pero para detectar errores, se debe enviar información adicional, por ejemplo, un campo con información redundante. Asimismo, debe haber un mecanismo a través del cual el receptor pueda notificar si hubo o no un error en la transmisión.

Por otra parte, es necesario identificar al emisor y receptor de la información en este nivel, ya sea porque el medio es compartido, o porque el mismo enlace (de datos), puede estar transportando flujos de información de distintas fuentes. Además, en algunas tecnologías de red, se debe indicar dónde comienza y dónde termina el PDU en esta capa (delimitadores de inicio y fin de trama. Es por ello que al datagrama que viene de la capa superior, se le debe añadir información adicional, representada por los campos *header* y *trailer* de la figura 3.2.

El armado de esta trama es la primer función de la capa de enlace de datos. Se añaden campos adicionales que iremos descubriendo a lo largo de este capítulo, y el datagrama de la capa de red es simplemente el **campo de datos** o carga útil de la trama.

3.2 Control de errores

En la actualidad la mayoría de los medios físicos cableados son muy confiables (en el caso de la fibra óptica, extremadamente confiables), mientras que por muy diversos factores, las comunicaciones inalámbricas suelen tener una probabilidad no despreciable de que las tramas se dañen (o corrompan).

En cualquier caso, es muy importante poder detectar que una trama se ha dañado. A esto se le conoce como *detección de errores*. Si la probabilidad de error es muy baja, como en las redes locales y en los enlaces de fibra óptica, una trama dañada simplemente se descarta, y se deja a las capas superiores (en particular, a la capa de transporte), que se ocupe de “recuperarla”.

En otros casos, como ocurría en las primeras redes de computadoras, correspondía a la capa de enlace de datos solicitar la retransmisión de una trama que llegó con error, o de una que simplemente no llegó. Así como puede ser que una trama no llegue (por ejemplo, el caso en el que en el nodo anterior el buffer estaba lleno y tuvo que ser descartada, o lo que se dañó fue el identificador del destinatario), puede ocurrir que la trama llegue duplicada (como veremos más adelante, esto suele ocurrir porque el emisor retransmite indebidamente una trama). Por estos casos, será necesario agregar un **número de secuencia** a las tramas con el cual nos será posible identificarlas.

3.2.1 Detección de errores

Para detectar un error lo que se hace es agregar a la trama información adicional generada tanto en el emisor como en el receptor. Con base en ello, el receptor puede determinar con alta probabilidad, si hubo o no un error en la transmisión. Hay muchas técnicas para ello y varían, justamente, en la precisión (la capacidad de no dejar pasar errores inadvertidos) y de la complejidad. En esta sección veremos algunos ejemplos.

Detección de paridad

La detección de errores por bit de paridad es sumamente conocida en computación pues así es como se solía verificar la integridad de las palabras de memoria RAM. También era popular para la transmisión de caracteres ASCII en algunos protocolos de comunicación.

La idea básica es añadir un bit llamado de paridad a cada octeto o palabra, de manera que la suma de todos los bits, incluyendo el de paridad, sea par (paridad par) o impar (paridad non). Por ejemplo, si el octeto tiene un valor 00110110 y se quiere verificar la integridad con paridad non, el bit de paridad tendría un valor de “1”.

Este es un mecanismo muy sencillo y de bajo costo, pero únicamente puede detectar un número impar de errores en una palabra. En comunicaciones esto no es apropiado, pues cuando hay errores por interferencia, éstos suelen afectar una secuencia de bits, por lo que este método tiene únicamente una fiabilidad de 50%.

Una pequeña mejora consiste en agrupar una secuencia de palabras a transmitir como una matriz, y calcular tanto la paridad de cada palabra, llamada VRC, *Vertical Redundancy Check*, como la de cada renglón, llamada LRC, *Longitudinal Redundancy Check*. En la tabla de la figura 3.3 se muestra un ejemplo de este método para la palabra “Hola”. Cada columna representa el código ASCII de la letra correspondiente y se está utilizando paridad non. En la transmisión se suele enviar cada letra con su bit de paridad y posteriormente el patrón correspondiente al valor LRC calculado, pero se permiten otras variantes siempre y cuando transmisor y receptor sigan las mismas reglas.

	H	o	l	a	LRC
	0	0	0	0	1
	1	1	1	1	1
	0	1	1	1	0
	0	0	0	0	1
	1	1	1	0	0
	0	1	1	0	1
	0	1	0	0	0
	0	1	0	1	1
VRC	1	1	1	0	0

Figura 3.3: *Ejemplo para la generación de bits verificadores LRC y VRC.*

La combinación VRC/LRC es un poco más confiable. Permite detectar errores en pequeñas secuencias y, en caso de errores individuales, hasta pueden corregirse automáticamente. El problema, en su aplicación práctica, es que no se puede saber si efectivamente ha habido error en un solo bit, o combinaciones de errores en parejas de bits, que pasan inadvertidas.

Checksum

La detección de errores por validación de suma o Checksum también es un método muy sencillo y un poco más confiable que la verificación por paridad VRC/LRC. Consiste en considerar todos los octetos en la trama como números binarios sin signo y calcular la suma típicamente en complemento a 1, lo que significa que si hay un acarreo de salida, se incrementa en uno el resultado. El valor resultante, que es de la misma longitud (un octeto o una palabra), se complementa a unos y se agrega al final de la trama.

El receptor hace una operación similar, incluyendo el campo de checksum recibido. El complemento a unos de la suma en el receptor debe ser cero; de lo contrario, se tiene la certeza de que la trama tuvo un error y debe ser descartada.

En la figura 3.4 se muestra un ejemplo del cálculo de checksum para el mensaje “Hola” en el transmisor, y la verificación en el receptor.

	Emisor									receptor							
H	0	1	0	0	1	0	0	0		0	1	0	0	1	0	0	0
o	0	1	1	0	1	1	1	1		0	1	1	0	1	1	1	1
	1	0	1	1	0	1	1	1		1	0	1	1	0	1	1	1
l	0	1	1	0	1	1	0	0		0	1	1	0	1	1	0	0
	10	0	1	0	0	0	1	1		10	0	1	0	0	0	1	1
								1									1
	0	0	1	0	0	1	0	0		0	0	1	0	0	1	0	0
a	0	1	1	0	0	0	0	1		0	1	1	0	0	0	0	1
Suma:	1	0	0	0	0	1	0	1		1	0	0	0	0	1	0	1
CheckSum:	0	1	1	1	1	0	1	0		0	1	1	1	1	0	1	0
									Suma:	1	1	1	1	1	1	1	1
									Complemento:	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

Figura 3.4: *Ejemplo de verificación de integridad mediante sumas en complemento a unos.*

El método de checksum se utiliza con cierta frecuencia para verificar la integridad de datos almacenados y en protocolos en la capa de aplicación debido a su simplicidad. Para las capas inferiores no es un método recomendable porque sigue siendo poco confiable: combinaciones de errores en distintas palabras pueden producir el mismo patrón de verificación.

Códigos cíclicos redundantes

En este método, al igual que en el caso anterior, se agregan bits de redundancia al final de la secuencia de datos a transmitir. Estos bits se generan como el remanente de una división binaria utilizando aritmética módulo 2.

Si se agregó el remanente de una división a la secuencia, ésta es exactamente divisible por el divisor. Eso es lo que hace el receptor: divide el mensaje que llega, formado por la secuencia más el código de verificación (los bits de redundancia) y si el remanente es cero, se tiene una muy alta probabilidad de que el mensaje llegó sin error.

El método funciona de la siguiente manera:

- D es el mensaje original, por ejemplo $D = 11010100$. Se va a representar como un polinomio $D(x)$ con coeficientes 0 y 1:
  $D(x) = 1 \times x^7 + 1 \times x^6 + 0 \times x^5 + 1 \times x^4 + 0 \times x^3 + 1 \times x^2 + 0 \times x^1 + 0 \times x^0$
- G(x) es el divisor, conocido como el polinomio generador. Usemos como ejemplo $G(x) = 1 \times x^3 + 0 \times x^2 + 1 \times x^1 + 1 \times x^0$, es decir, $G = 1011$. El grado de este polinomio es $r = 3$.
- La secuencia original se multiplica por 2^r, lo que equivale a agregar r ceros a la derecha: $D' = 11010100000$
- Se divide D' por $G(x)$ utilizando aritmética módulo 2. En la aritmética módulo 2, no hay acarreo ni préstamo; la suma y la resta son idénticas y equivalen a la operación binaria XOR. El verificador del mensaje es el remanente $E(x)$.
- Se transmite el mensaje original junto con el remanente $E(x)$
- En el receptor, se divide la secuencia recibida entre $G(x)$ y se revisa que el remanente sea cero. De no ser así, la trama debe ser descartada.

La figura 3.5 muestra conceptualmente la generación del código verificador en el transmisor, del lado izquierdo, y la verificación de la trama en el receptor del lado derecho para los valores de ejemplo que hemos usado en esta sección.

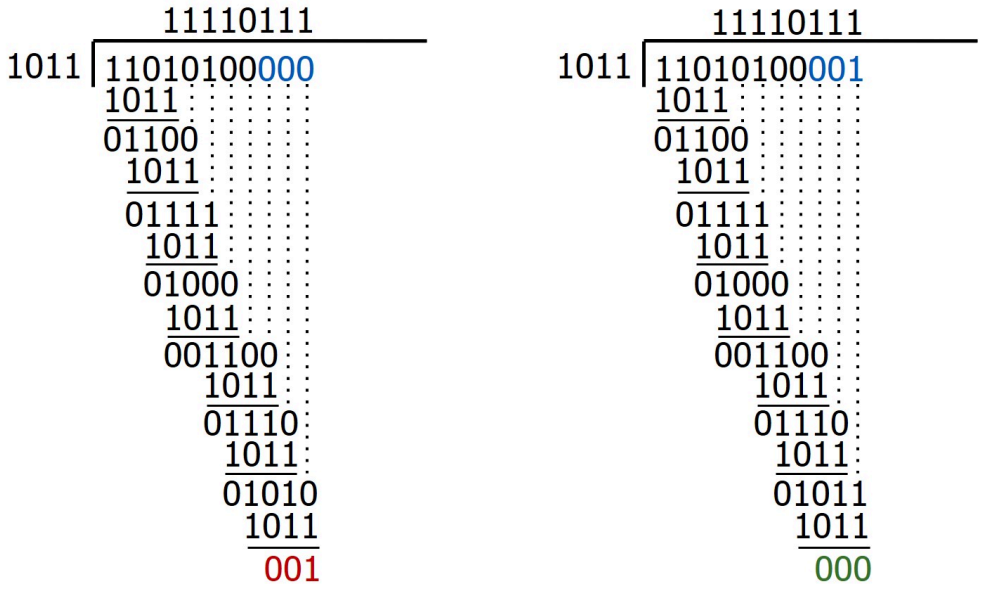

Figura 3.5: *Ejemplo de generación de bits redundantes con CRC (izquierda) y verificación de integridad (derecha).*

Los métodos de CRC para detección de errores son, por mucho, los más utilizados en redes de computadoras. En primer lugar, tienen excelentes propiedades para detección de errores. Un CRC con un polinomio generador de grado r detecta:

- Todos los errores de un bit
- Casi todos los errores de dos bits
- Cualquier número impar de errores
- Todos los errores que ocurren en ráfagas $\leq r$ bits

Lo más sorprendente, es que si la transmisión de datos es en serie, como ocurre en prácticamente todas las redes, el circuito para generar y verificar la trama es muy simple. Dado que la resta en la división es una operación XOR y el polinomio generador se va “restando” secuencialmente de la trama de datos, el circuito está conformado por un registro de corrimientos con r flip-flops D, con compuertas XOR en los elementos que corresponden a coeficientes igual a 1 en el generador. Para nuestro generador $G = 1011$, el circuito es el de la figura 3.6.

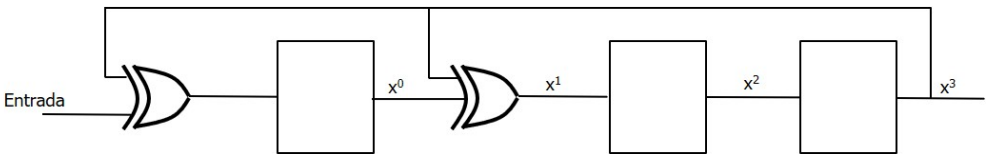

Figura 3.6: *Registro de corrimientos para el Generador* $G(x) = x^3 + x^1 + 1$.

En el transmisor los flip-flops se inicializan con cero. La secuencia de datos D se introduce al registro de corrimientos y se envía por el canal. Al final, el patrón que queda en el registro de corrimientos corresponde al remanente y es el que se envía junto con la secuencia.

En el receptor, el registro también se inicializa con cero; la trama se introduce al registro bit a bit, incluyendo los bits de redundancia. Al final, si no hubo error en la trama, los flip-flops del registro deben contener cero.

La tabla 3{reference-type="ref" @ref(t:polpop"} muestra algunos de los polinomios generadores más populares en redes de computadoras.

Algunos polinomios generadores bien conocidos

CRC	G(X)
CRC-8	$X^8 + X^2 + X^1 + 1$
CRC-10	$X^{10} + X^9 + X^5 + X^4 + X^1 + 1$
CRC-12	$X^{12} + X^{11} + X^3 + X^2 + X^1 + 1$
CRC-16	$X^{16} + X^{15} + X^2 + 1$
CRC-CCITT	$X^{16} + X^{12} + X^5 + 1$
CRC-32	$X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} +$
	$X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X^1 + 1$

3.2.2 Auto-corrección de errores

Como se mostrará más adelante, en ocasiones es sumamente ineficiente (y a veces imposible) solicitar la retransmisión de una trama que ha llegado con error. Para estos casos, se han propuesto códigos auto-correctores basados en teoría de la información que añaden redundancia a la información de manera tal que son capaces de corregir automáticamente algunos de los patrones de errores que ocurren con mayor frecuencia.

Un ejemplo es la codificación Reed-Solomon, muy utilizada para proteger las pistas de música en los discos compactos (CD). Si hay un daño (pequeño) en la superficie del disco, la redundancia añadida al grabar el disco permite reproducir la canción con bastante fidelidad.

En redes de comunicaciones a este tipo de esquemas se les conoce como FEC, *Forward Error Correction*. Uno de los primeros, y de los más sencillos es la codificación de Hamming, que es la que presentaremos brevemente. Mecanismos FEC mucho más poderosos son los códigos Viterbi o los utilizados en los llamados turbo-códigos, basados códigos convolucionales, que se emplean en las comunicaciones espaciales modernas. Su complejidad rebasa el alcance de estas notas.

La idea básica es utilizar más bits de los estrictamente necesarios para codificar un "símbolo". Por ejemplo, si deseamos codificar un "alfabeto" con cuatro letras, se necesitan únicamente 2 bits, pues $2^2 = 4$, pero podríamos codificar cada una de esas letras con un patrón de, digamos, 10 bits.

Se le llama **distancia de Hamming, d,** al número de bits en que difieren dos palabras del mismo código y se obtiene simplemente al aplicar el XOR entre las dos palabras y contar el número de unos. Si en nuestro ejemplo se eligen códigos adecuados para las cuatro letras, se necesitan d errores para que el código de una letra se "transforme" en el código de otra, generando un error en el mensaje.

En términos generales si una palabra tiene m bits, se pueden tener 2^m combinaciones o mensajes (en el ejemplo del párrafo anterior, con $m = 2$ se puede tener un patrón distinto para cada una de las cuatro letras de nuestro alfabeto). Si queremos ser capaces de detectar y *corregir* errores en el mensaje (en la codificación de una letra), debemos agregar r bits de redundancia, de manera que cada mensaje está codificado con un patrón de $n = m + r$ bits.

Para poder corregir **UN error**, cada uno de los 2^m debe tener asociado n palabras que difieren de él en un solo bit y, por supuesto, que no forman otro mensaje de nuestro "alfabeto". Entonces tenemos (para un bit):

$$
\begin{aligned}
(n+1) \times 2^m &\leq 2^n \#\text{n palabras distintas} + 1 \text{ correcta} \\
(m+r+1) \times 2^m &\leq 2^{m+r} \\
2^{m+r} &\geq (m+r+1) \times 2^m \\
2^r &\geq m+r+1
\end{aligned}
$$

En particular, si se desea proteger caracteres codificados en ASCII sin extensiones que tienen un patrón de 7 bits, necesitamos añadir 4 bits de redundancia: $2^4 = 16 \geq (7+4+1) = 12$.

Código de Hamming

En 1950 Richard Hamming ideó un ingenioso método posicional para detectar y corregir errores en las tarjetas perforadas que se utilizaban para introducir código y datos en las computadoras de aquella época. El método es el siguiente:

1. Se calcula el número de bits de redundancia necesarios para proteger los mensajes, de acuerdo a la ecuación anterior. En el ejemplo que sigue, utilizaremos mensajes de seis bits, por lo que necesitaremos $r = 4$ bits de redundancia.

2. En las secuencias codificadas, se reservan las posiciones de las potencias de 2 para los bits de verificación. En nuestro ejemplo, se reservan las posiciones $2^0 = 1, 2^1 = 2, 2^2 = 4, 2^3 = 8$ para los bits redundantes. Nuestras secuencias tendrán 10 bits: 6 del mensaje más 4 de los códigos de verificación.

3. El valor de los bits de verificación se determina calculando la paridad de los bits de datos en función de su posición, descompuesta en valor binario. Por ejemplo, el primer bit del mensaje está en la tercera posición (las primeras dos están ocupadas por bits de verificación) y por tanto será utilizado para calcular la paridad de los bits de verificación en las posiciones 1 y 2, dado que $3 = 2^0 + 2^1 = 1 + 2 \rightarrow 1, 2$.

   De forma similar, el quinto bit del mensaje, que se encontrará en la posición 9 del código, contribuye a calcular la paridad de los bits de verificación en las posiciones 1 y 8: $9 = 2^0 + 2^3 = 1 + 8 \rightarrow 1, 8$.

4. En el receptor se sigue un procedimiento similar y se aplica la operación XOR entre los bits de redundancia recibidos y los calculados. Si el resultado es cero, no hubo error. Si es distinto de cero, indica qué bit es el incorrecto y se puede corregir automáticamente. Una observación: para determinar la posición del bit en decimal, se debe invertir el orden de los bits.

La figura 3.7 muestra un ejemplo de cómo se calculan los bits de redundancia al transmitir un mensaje con tres palabras de 6 bits cada una (pasos 1 a 4). Por ejemplo, el bit de verificación en la posición 2 se calcula con la paridad non de los bits de datos en las posiciones 3, 6, 7 y 10).

La tabla 5 ejemplifica un bit erróneo (en color morado) en cada palabra y los bits de verificación resultantes. Se resaltan en verde los bits de verificación que cambiaron.

Finalmente, al aplicar el XOR entre los bits de verificación recibidos y los calculados, se obtiene la posición del bit erróneo en cada palabra.

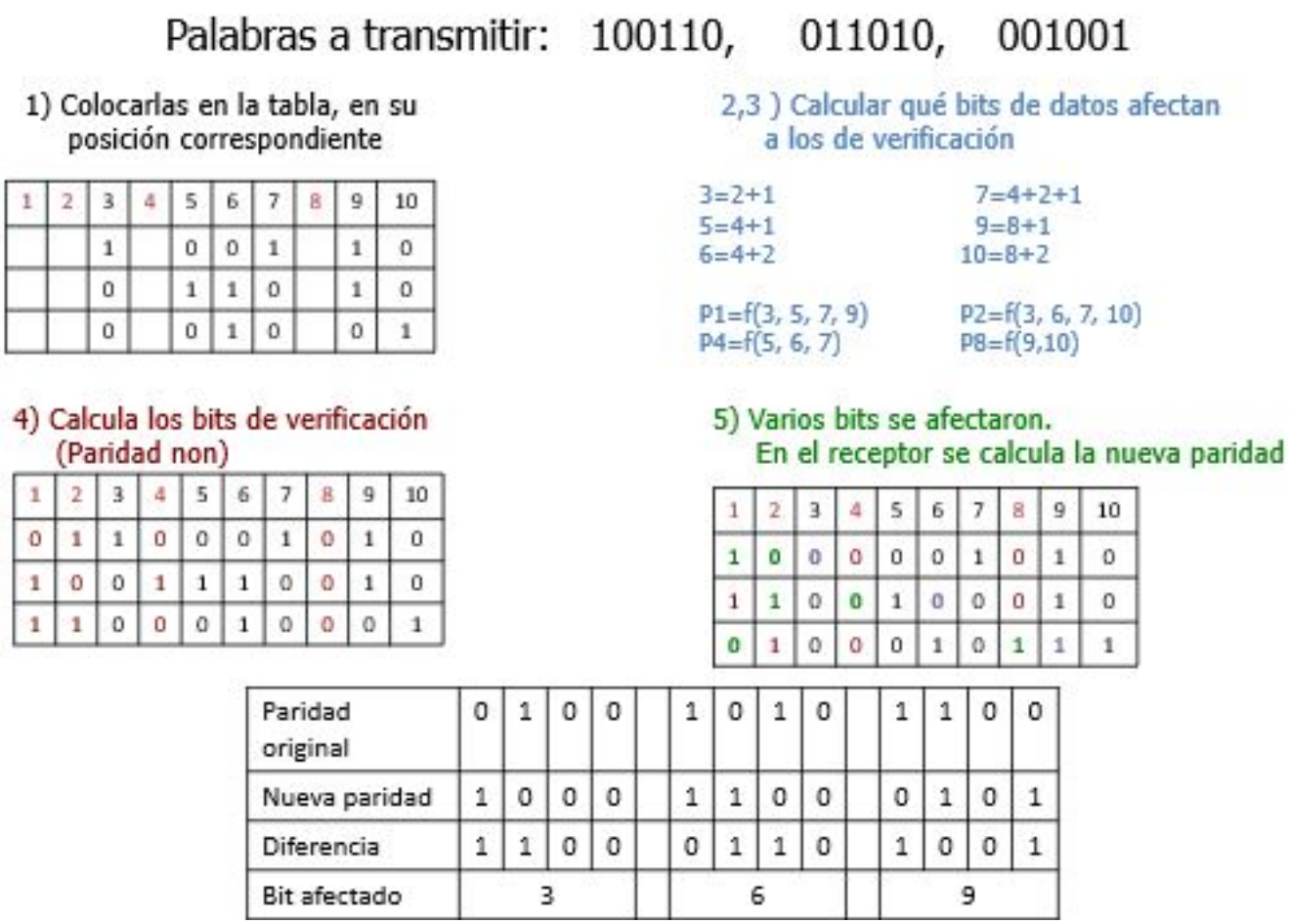

Figura 3.7: *Ejemplo para calcular los bits de verificación utilizando códigos de hamming.*

3.3 Protocolos para corrección de errores

Los mecanismos de auto-corrección de errores agregan mucha redundancia y están muy limitados en la cantidad de errores que pueden corregir. En la práctica, sólo se utilizan cuando la retransmisión de la información dañada es imposible o muy poco práctica.

Lo más común en redes de comunicaciones, es utilizar mecanismos en los que el receptor notifica la correcta recepción de la trama a través de un acuse de recibo (PAR, *Positive Acknowledge Reception*). Una trama recibida con error o no recibida, activa automáticamente la retransmisión de la trama. Por eso, estos mecanismos se conocen genéricamente como **Protocolos ARQ, *Automatic Repeat reQuest***. Existen muchas variaciones de estos protocolos. En esta sección presentamos los conceptos básicos.

3.3.1 Protocolo Stop and Wait

El principio de operación del protocolo Stop and Wait (parada y espera), también llamado Send and Wait, se muestra en la figura 3.8. A la izquierda se encuentra el transmisor. La capa superior (en este caso, la capa de red) solicita transmitir un datagrama. La capa de enlace de datos calcula la secuencia de verificación de integridad (con un CheckSum o CRC), arma la trama (el PDU de datos) y la envía. Al mismo tiempo, activa un temporizador.

Un poco más tarde (el retardo de propagación se representa con la pequeña inclinación de la flecha) la trama llega al destinatario. Éste verifica que la trama ha llegado correctamente y envía de regreso un acuse de recibo positivo (ACK). Al mismo tiempo, notifica a la capa superior de que tiene datos para ella.

Mientras se está en espera del acuse, la capa superior solicita el envío de otro datagrama. Esta solicitud se retiene hasta que llegue el acuse de la trama anterior. Por eso el protocolo se llama Stop and Wait: envía la trama y se detiene en espera del acuse de recibo.

La siguiente trama se pierde o sufre alguna perturbación y llega al destino con información incorrecta. En esta implementación del protocolo, la trama es descartada y NO se envía el acuse de recibo. Esa es la función del temporizador: Si cuando éste se agota no se ha recibido el acuse, la trama se reenvía automáticamente (ARQ).

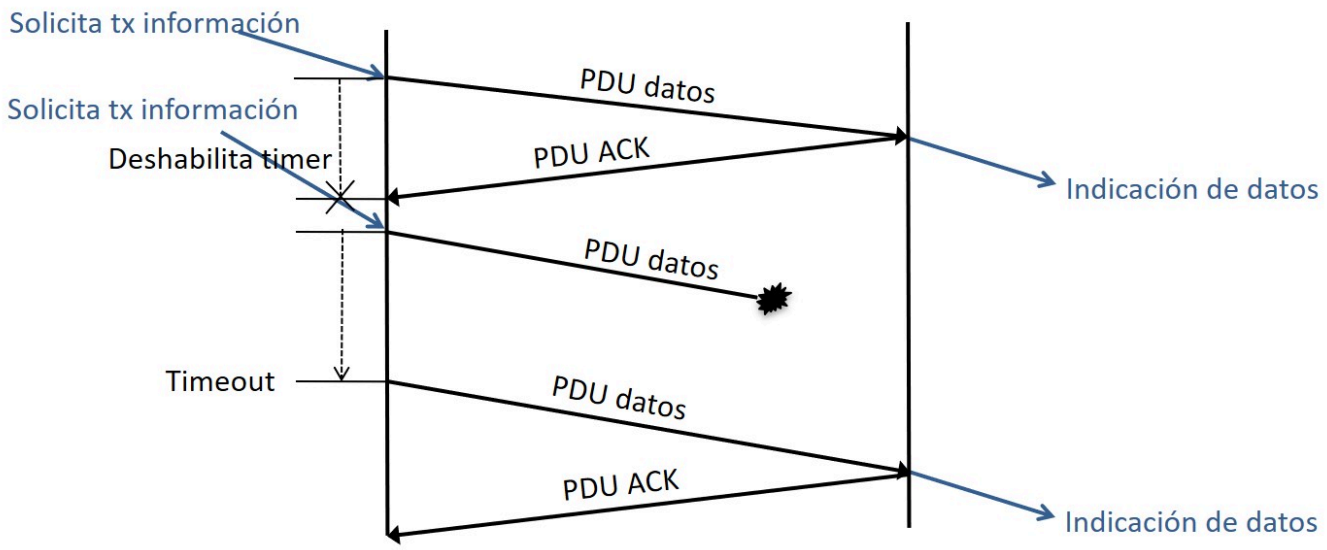

Figura 3.8: *Protocolo Stop and Wait. Una trama dañada.*

Si la trama llega incorrecta, el receptor podría enviar un acuse notificando el error, lo que se conoce como Acuse de Recibo Negativo, **NACK**. Varios protocolos implementan este mecanismo y, de hecho, mejora la eficiencia de la comunicación al no tener que esperar que se

active el temporizador. Sin embargo, su uso es poco común pues se argumenta que aumenta mucho la complejidad de los protocolos y ésta no se justifica si los errores son relativamente raros.

Es posible que la trama llegó correctamente pero su acuse se dañó o se perdió, como se muestra en la figura 3.9 (la figura está simplificada para mayor claridad). En ese caso, al agotarse el temporizador, el emisor enviará nuevamente la trama, pero para el receptor esta es una trama duplicada que debe descartarse.

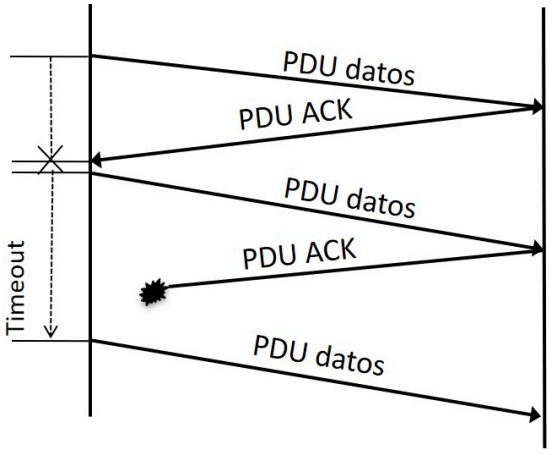

Figura 3.9: *Acuse de recibo dañado.*

3.3.2 Protocolo de bit alternado

Para resolver el problema de recepción de tramas duplicadas, lo que debe hacerse es enumerar las tramas y los acuses de recibo agregando un campo llamado **Número de Secuencia**. Como solamente hay una trama en tránsito (es decir, en camino del emisor al receptor) y no se libera otra hasta no recibir su acuse, el número de secuencia puede ser de un solo bit en el envío de las tramas y de sus acuses de recibo.

En la figura 3.10 se observa la operación del protocolo de bit alternado. El cronograma se coloca de forma horizontal para ahorrar espacio. Se observa cómo se van alternando los números de secuencia 0 y 1. Con el fin de hacer más clara la asignación de números de secuencia, se han coloreado las tramas con número de secuencia “0” y sus acuses de recibo en azul, y aquéllas con números de secuencia “1” y sus ACK en naranja.

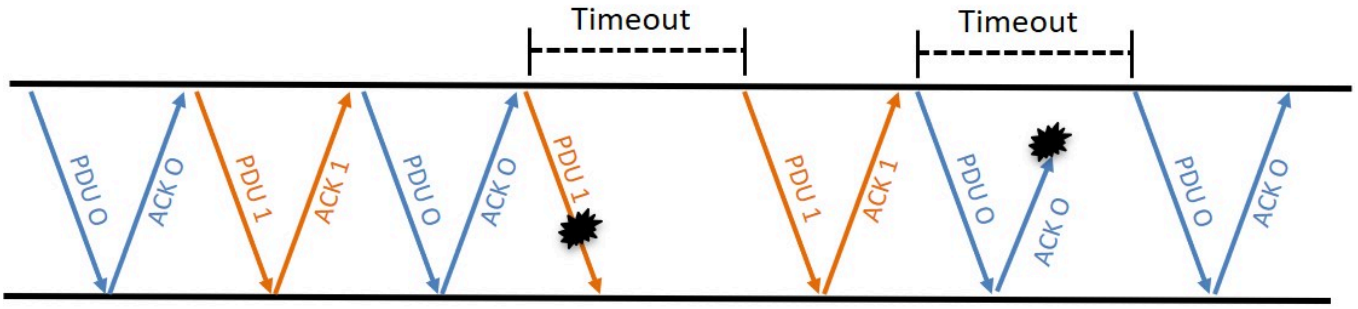

Figura 3.10: *Protocolo de bit alternado.*

La figura muestra cómo el protocolo se recupera automáticamente (ARQ) cuando una trama llega dañada (la tercer trama en la figura) y el receptor la descarta sin enviar el acuse de recibo, así como cuando lo que se pierde (o descarta) es un acuse de recibo. En ambos casos, la activación del temporizador reenvía una trama. En el último ejemplo, gracias al número de secuencia (con valor 0 en la figura), el receptor se da cuenta de que es una trama duplicada; la descarta pero envía el acuse de recibo.

Es muy importante que el valor de los temporizadores esté bien ajustado. Si el temporizador es muy corto, se reenviarán tramas innecesariamente, como se muestra en la imagen izquierda de la figura 3.11, mientras que si es muy largo (imagen derecha), se pierde mucho tiempo en retransmitir la trama. En ambos casos, el desempeño de la red (un término que definiremos formalmente más adelante), se degrada severamente.

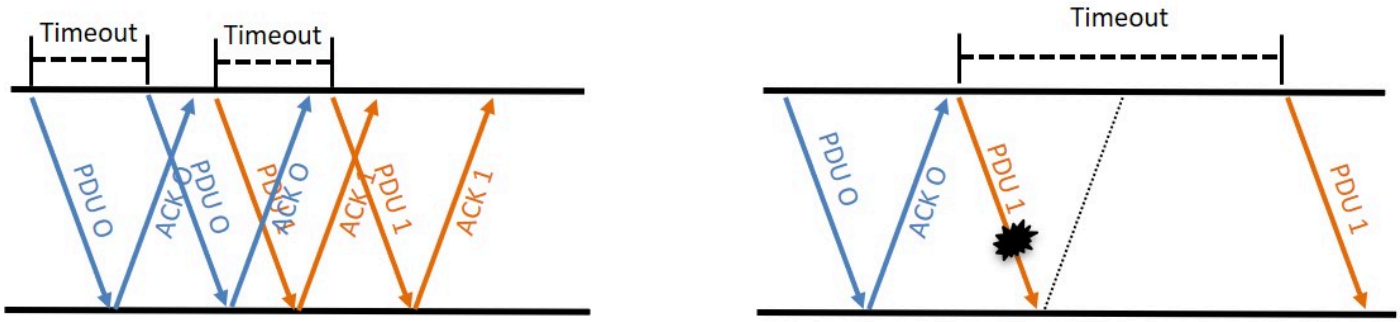

Figura 3.11: *Efecto de temporizadores mal ajustados.*

El valor ideal del temporizador es ligeramente superior al retardo de ida y vuelta (RTT, *round trip time*) que experimentan la trama y su acuse de recibo. Dado que este retardo no se conoce a priori y además puede variar, este tipo de temporizadores suele ajustarse dinámicamente a las condiciones de la red. En el capítulo correspondiente a la capa de transporte veremos cómo lo hace el protocolo TCP.

El protocolo de bit alternado es el primer protocolo propiamente de red que hemos visto en el curso. De una manera simple e ingeniosa, busca ofrecer un servicio de entrega confiable de datagramas. Sin embargo, su simplicidad viene a expensas de una pésima eficiencia.

Ejemplo 3.1 Considere un enlace de fibra óptica entre dos ciudades a 100 kilómetros de distancia. Sin tomar en cuenta ninguna otra fuente de retraso, el retardo de ida y vuelta en este enlace sería de $t_{RTT} = (2 \times 100\,\mathrm{km})/200,000\,\mathrm{km/s} = 1\,\mathrm{ms}$.

Si la trama es de 8192 bits (1 kByte, lo cual es bastante realista) y la velocidad del enlace es de 100 Mb/s, el retardo de transmisión (es decir, el tiempo que toma "colocar" la trama en la red es $t_t = 8,192\,\mathrm{b}/100,000,000\,\mathrm{b/s} \approx 0.08\,\mathrm{ms}$. Es decir, el enlace está ocioso esperando un acuse de recibo cuando se podrían estar enviando un poco más de 10 tramas.

3.3.3 Protocolos de ventanas deslizantes

Analizaremos el comportamiento de estos protocolos con mucho mayor detalle en el capítulo correspondiente a la capa de transporte. En esta sección nos limitaremos a presentar los conceptos básicos.

El objetivo principal de estos protocolos es mejorar la eficiencia al permitir que varias tramas sean transmitidas mientras se espera el acuse de recibo. El emisor puede transmitir las tramas que se encuentran dentro de un rango o “ventana”, como la mostrada en azul en la figura 3.12.

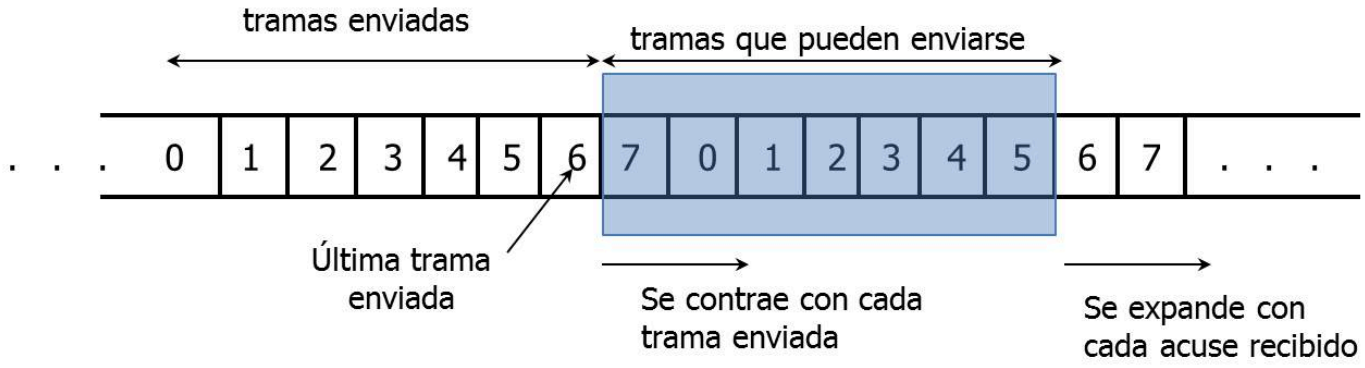

Figura 3.12: *Ventana deslizante en el transmisor.*

Con cada trama enviada, la ventana se “cierra” una posición. En nuestra figura, la frontera izquierda de la ventana se recorre a la derecha, de la posición 6 a la 7, reduciendo su tamaño en uno. Con cada acuse recibido, la ventana se “abre” una posición. En nuestra figura, la frontera derecha de la ventana se recorre a la derecha, de la posición 5 a la 6, ampliando su tamaño en uno. Así es como la ventana se va “deslizando” por el conjunto (conceptual) de tramas a ser emitidas.

El tamaño de la ventana depende de muchos factores que, lamentablemente, no podremos analizar por ahora. El tamaño de la ventana determina los números de secuencia que, como en el protocolo del bit alternado, pueden ser reutilizados. En la figura 3.12, el tamaño de la ventana es de 8 tramas (y por consiguiente, el campo que identifica al número de trama, es de 3 bits).

En estricto sentido, las variantes más sencillas de esta familia de protocolos no requiere de una ventana del lado del receptor. Sin embargo, es muy común que ésta se implemente como un buffer en el que se almacenan temporalmente las tramas hasta que puedan ser entregadas a la capa superior[17]. Por ello, en las figuras de esta sección estaremos mostrando ventanas tanto en el transmisor como en el receptor.

En la figura 3.13 se ejemplifica la operación de este protocolo con un tamaño de ventana de cuatro tramas.

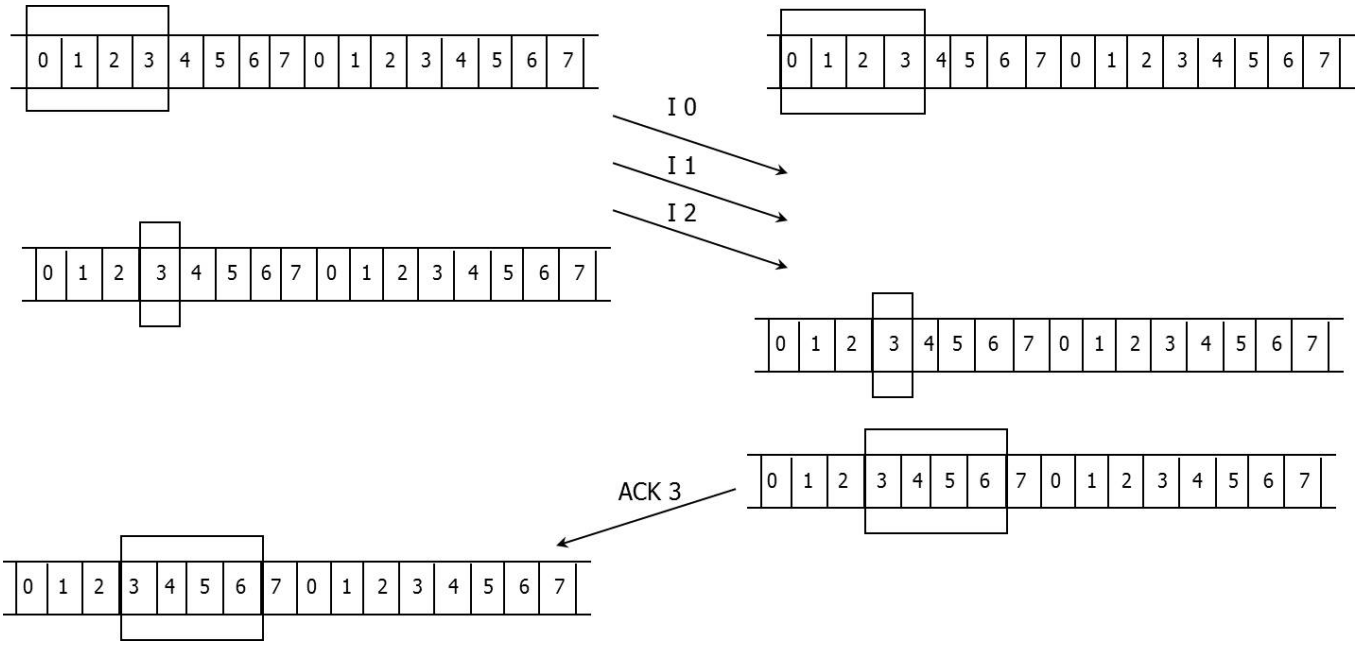

Figura 3.13: *Dinámica del protocolo de ventanas deslizantes.*

- Inicialmente las ventanas están completamente abiertas, es decir, el transmisor puede enviar cuatro tramas y el receptor tiene capacidad par aceptarlas todas.
- Se envían las tramas 0, 1, 2. La ventana del emisor se reduce a 1, al igual que la ventana del receptor cuando éstas llegan a su destino.
- Cuando el receptor verifica la integridad de las tramas y puede entregarlas a la capa superior, envía el acuse de recibo, con el cual la ventana del transmisor (y la suya) se expanden nuevamente.

  Hay dos cosas a resaltar en el acuse de recibo mostrado en la figura:

1. Los acuses de recibo positivos suelen indicar siguiente trama que se espera recibir, no la trama recibida correctamente. El indicador ACK 3 se interpreta como que el receptor espera que la siguiente trama a recibir tendrá un número de secuencia 3, señalando implícitamente que la última trama recibida correctamente, fue la 2.

2. No es necesario, y en realidad es indeseable, enviar un acuse por cada trama recibida correctamente. Con el acuse de recibo N, se da por sentado que llegaron correctamente hasta la trama $N-1$.

Rechazos

Existen básicamente dos maneras de tratar el caso en que alguna de las tramas emitidas en una ventana sufre alguna alteración y llega con error: Regreso a N (*Go Back to N*) y acuses selectivos (*Selective Acknowledgment*).

Supongamos que se envía una ráfaga de siete tramas con los números de secuencia 0 a 6 y la trama 4 llega con error. En Go Back to N se retransmiten las tramas 4 en adelante, independientemente de que las tramas 5 y 6 hayan llegado correctamente al destino. En rechazos selectivos, sólo se pide la retransmisión de la trama 4; la 5 y 6 quedan almacenadas en el buffer del receptor que no las entregará a la capa superior hasta que la trama 4 haya sido reenviada y recibida correctamente.

Por supuesto, el mecanismo de acuses selectivos permite comunicaciones más eficientes pero es mucho más complejo y en la práctica se utiliza relativamente poco. Es mucho más frecuente encontrar protocolos Go Back to N en las redes contemporáneas.

3.4 Problemas

Problema 3.1 Un protocolo de capa 2 puede transportar hasta 15 paquetes en una trama y soporta acuses de recibo en {} Cada paquete puede ser de hasta 64 bytes. La trama se protege con un algoritmo CRC-16.

Muestre un posible formato de trama indicando el tamaño aproximado de cada campo.

Problema 3.2 Para el mensaje “ITAM” (códigos ASCII 49H; 54H; 41H; 4DH), calcule el campo de verificación de integridad con Checksum

Problema 3.3 En un mecanismo de chequeo cíclico redundante al divisor se le llama:

- Grado
- CRC
- Generador
- Matriz

Problema 3.4 ¿Cuál es la distancia de Hamming de los siguientes pares de palabras: (0010,0001); (1010, 0110); (1111,0011)

Problema 3.5 Explique muy brevemente qué protocolo es más eficiente Stop and wait, Go Back N o de rechazo selectivo.

Problema 3.6 Dos nodos vecinos, A y B utilizan un protocolo de de ventanas deslizantes con una ventana de cuatro paquetes de tamaño y mecanismo de control Go Back N. Suponga que A transmite y B recibe. Muestre las posiciones de la ventana para la siguiente sucesión de eventos:

- Antes de que se transmita cualquier paquete
- Después de que A ha enviado las tramas 0, 1, 2 y se han recibido los acuses de recibo de 0 y 1
- Después de que A ha enviado las tramas 3, 4, y 5 y se ha recibido el acuse de recibo de la 4.

Problema 3.7 Un enlace tiene una capacidad de 2 Mb/s y un retardo de propagación (en un sentido) de 20 mSeg. Si el tamaño de una trama es de 512 Bytes,

¿Cuál es el mínimo tamaño de ventana (es decir, de tramas en tránsito) para que el enlace tenga una eficiencia de al menos 50%? (Ignore el tamaño de los acuses de recibo)

¿Cuál es el valor ideal del temporizador para que ni haya retransmisiones innecesarias ni se pierda demasiado tiempo esperando el acuse de recibo?

Problema 3.8 ¿Qué mecanismo limita la cantidad de información que puede ser emitida sin esperar un acuse de recibo?

- Control de flujo
- Control de errores
- Control de congestión
- Control de acceso

17. Este buffer es necesario para implementar control de flujo y para la variante de acuses de recibo selectivos, temas que serán tratados en el capítulo correspondiente a la capa de transporte. ↩