Comunicaciones de Datos

Facultad de Ciencias Exactas y Naturales y Agrimensura.
Universidad Nacional del Nordeste

Guía Serie de Trabajos Prácticos Nº 2

Códigos Detectores y Correctores de Errores

1. Introducción

El objetivo principal en el diseño de redes de comunicaciones digitales es enviar información de manera confiable y eficiente entre nodos.

Los datos transmitidos se pueden corromper durante la transmisión.

En los sistemas de transmisión digital se dice que ha habido un error cuando se altera un bit.

Los errores aislados alteran un único bit. Se dice que ha habido una ráfaga de longitud B cuando se recibe una secuencia de B bits en la que son erróneos el primero, el último y cualquier número de bits intermedios.

El concepto central en la detección o corrección de errores es la **redundancia**. Para poder detectar o corregir errores, es necesario enviar algunos bits adicionales junto con los bits de datos.

La **detección de errores**, solo mira si ha ocurrido algún error, da lo mismo un error aislado o una ráfaga de longitud B. La **corrección de errores**, por el contrario, necesita saber el número de bits dañados y su posición dentro del mensaje.

En la codificación de bloques, se divide el mensaje en bloques de k bits, denominados **palabras de datos**. Se añaden r bits redundantes a cada bloque hasta conseguir bloques con una longitud de $n = k + r$ bits, denominados **palabras código.** Más adelante, describiremos cómo se eligen o calculan los bits de redundancia. Es importante notar que, de esta manera, se tiene un conjunto con 2^k palabras de datos y un conjunto de 2^n palabras código. Puesto que $n > k$, el número posible de palabras código es mayor que el número posible de palabras de datos. El proceso de codificación de bloques es uno a uno, esto significa que se tienen $2^n - 2^k$ palabras código que no se utilizan denominados códigos ilegales o no válidos.

Si se quiere detectar errores, se debe diseñar un código de tal forma que si a una palabra del código se le cambia un único símbolo, la palabra resultante no sea una palabra del código para así poder saber que se ha producido un error.

Si además se quiere corregir errores, hay que saber cuál es la palabra enviada. La idea básica es comparar la palabra recibida con todas las palabras del código y asignarle la palabra que difiera en menos símbolos.

Supongamos el siguiente conjunto de mensajes, codificados con palabras de datos de $k = 2$ bits.

$$C = \{00; 01; 10; 11\}$$

¿Cómo se pueden detectar errores utilizando bits de redundancia?

En la Tabla 1 se presenta un ejemplo de código para la detección de errores simples, añadiendo a las palabras de datos un bit de redundancia (bit de paridad) con lo que se obtienen palabras código con $n = 3$ bits. El bit de paridad se determina de tal forma que el código tenga un número par de unos (paridad par).

Tabla 1. Un código para detectar errores simples

palabra datos	palabra código
00	000
01	011
10	101
11	110

El emisor codifica la palabra de datos 01 como 011 y la envía al receptor. Consideremos los siguientes casos:

1. El receptor recibe 011, es una palabra código válida. El receptor recupera la palabra de datos 01.
2. El código se daña durante la transmisión y se recibe 111 (se daña el bit situado a la izquierda, el bit más significativo o MSB). El código recibido no es válido y se descarta. Si el receptor compara la palabra recibida con todas las palabras del código, encuentra dos palabras (110 y 011) que difieren con esta en un símbolo.
3. El código se daña durante la transmisión y se recibe 000 (se dañan los dos bits de la derecha). El código recibido es válido, el receptor recuperar la palabra 00. Sin embargo, errores en dos bits han hecho que el error sea indetectable.

Un código de detección de errores solo puede detectar la cantidad de errores para los cuales ha sido diseñado.

¿Cómo se pueden corregir errores utilizando bits de redundancia?

Para detectar errores es necesario modificar el código para conseguir que las palabras del mismo se parezcan menos entre sí.

En la Tabla 2 se presenta un ejemplo de código para la corrección de errores simples, añadiendo a las palabras de datos tres bit de redundancia con lo que se obtienen palabras código con $n = 5$ bits.

Tabla 2. Un código para corregir errores simples

palabra datos	palabra código
00	00000
01	01011
10	10101
11	11110

El emisor codifica la palabra de datos 01 con el código 01011 y la envía. El código se daña durante la transmisión y se recibe 01001 (un error en el segundo bit de la derecha). En primer lugar, el receptor encuentra que el código no es válido, esto significa que ha ocurrido un error (el primer paso es la detección de error). Asumiendo que solo se ha dañado un bit, el receptor:

1. Compara (Tabla 3) el código recibido con todos los códigos de la tabla y buscando la palabra más próxima (la palabra que difiere en menos símbolos).
2. Encuentra que el código válido debe ser 01011, que difiere con el código recibido en un único bit siendo la palabra más próxima, y lo reemplaza como palabra válida para finalmente recuperar la palabra de datos 01.

Tabla 3. Comparación de la palabra recibida con las palabras del código

00000 01001	difiere en dos símbolos
01011 01001	difiere en un símbolo
10101 01001	difiere en tres símbolos
11110 01001	difiere en cuatro símbolos

2. Distancia Hamming

Es uno de los conceptos fundamentales en la codificación para el control de errores. La distancia Hamming entre dos palabras x e y (del mismo tamaño) es el número de bits en que difieren y se indica como $d(x, y)$. Se puede calcular aplicando la operación $XOR(\oplus)$ o suma módulo 2 (Tabla 4) sobre las dos palabras y contar el número de unos resultantes.

Tabla 4. Operación $XOR(\oplus)$ o suma módulo 2

$\oplus$	1	0
1	0	1
0	1	0

Ejemplo 1. Calcular la distancia Hamming entre $x = 001$ e $y = 011$

Si aplicamos la operación XOR sobre las dos palabras y contamos el número de unos resultantes:

$$\begin{array}{rccc} & 0 & 0 & 1 \\ \oplus & 0 & 1 & 1 \\ \hline & 0 & 1 & 0 \end{array}$$

Obtenemos la distancia Hamming entre las palabras x e y:

$$d(001; 011) = 1$$

3. Mínima distancia Haming

Es la medida que se utiliza para diseñar un código. Se llama distancia mínima (o distancia) de un código C a:

$$d(C) = \min\{d(u, v)/u, v \in C, u \neq v\}$$

Es decir, la distancia del código es la mínima distancia Haming entre todos los pares de palabras diferentes del código.

Ejemplo 2. Obtener la distancia del código $C = \{00000; 01011; 10101; 11110\}$

Calculamos la distancia Hamming entre todos los pares posibles de palabras diferentes del código:

$$d(00000; 01011) = 3$$
$$d(00000; 10101) = 3$$
$$d(00000; 11110) = 4$$
$$d(01011; 10101) = 4$$
$$d(01011; 11110) = 3$$
$$d(10101; 11110) = 3$$

La distancia del código es, la mínima distancia Hamming:

$$d(C) = 3$$

Si se quiere ser capaz de detectar hasta s errores, entonces se debe diseñar un código con:

$$d(C) = s + 1$$

Si se quiere ser capaz de corregir hasta s errores, entonces se debe diseñar un código con:

$$d(C) = 2 \times s + 1$$

4. Códigos de bloques lineales

La mayoría de los códigos utilizados pertenecen a un subconjunto denominado *códigos de bloques lineales.*

Los códigos de bloques lineales toman un conjunto de palabras de datos de k bits (2^k mensajes) y generan un conjunto de palabras código de n bits (2^k códigos, con $n \geq k$) utilizando operaciones algebraicas sobre el bloque.

Informalmente, un código de bloques lineal es uno en el cual la operación $XOR(\oplus)$ entre dos palabras cualesquiera del código crea otra palabra código válida.

En general se utiliza la notación (n, k, d) código, donde los k bits de la palabra de datos se combinan para generar una palabra código de n bits (de manera que cada palabra código tiene $n - k$ bits de redundancia) y d se refiere la distancia del código.

La tasa de un código de bloques lineal se define como k/n; cuando mayor sea, menor será la redundancia del código.

En un código de bloques lineal, la suma de dos palabras código cualesquiera, es otra palabra del código y la palabra código 000…0, es una palabra del código.

Los cálculos en el caso de los códigos binarios utilizan aritmética módulo 2 que define las reglas para la suma y la multiplicación. Las reglas para la suma son vistas anteriormente en la Tabla 4. Las reglas para la multiplicación están definidas por la Tabla 5.

Tabla 5. Multiplicación $(*)$ en módulo 2

$*$	1	0
1	1	0
0	0	0

Ejemplo 3. El código de bloques $C = \{000; 101, 011\}$ no es un código de bloques lineal, puesto que $101 \oplus 011 = 110$ no es una palabra del código. Agregando esta palabra al código, obtenemos un código de bloques lineal, puesto que la suma de cualesquiera dos palabras código es otra palabra código.

5. Códigos de Hamming

Los códigos de Hamming fueron originalmente diseñados con una distancia $d(C) = 3$, lo que significa que pueden detectar hasta dos errores y corregir errores simples.

En estos códigos, los bits de redundancia (o paridad, obtenidos a partir de los bits de datos), ocupan posiciones que son potencia de dos (1,2,4,8,...) el resto de las posiciones (3, 5, 6,7, 9, 10,…) se completan con los bits de datos.

El (7,4,3) código de Hamming, utiliza 3 bits de paridad para proteger 4 bits de datos, como se describe a continuación.

Dada una palabra de datos de 4 bits:

$$b_4 b_3 b_2 b_1$$

En el transmisor se introducen 3 bits de paridad:

$$p_3 p_2 p_1$$

Para forma una palabra código de 7 bits, donde los bits de paridad ocupan posiciones que son potencia de dos:

b_4	b_3	b_2	$\boldsymbol{p_3}$	b_1	$\boldsymbol{p_2}$	$\boldsymbol{p_1}$
7	6	5	**4**	3	**2**	**1**
			$\mathbf{2^2}$		$\mathbf{2^1}$	$\mathbf{2^0}$

Los bits de paridad se calculan mediante combinaciones lineales de los cuatro bits de datos, para obtener lo que se conoce como ecuaciones para el cálculo de los bits de paridad:

$$\begin{cases} p_3 = b_2 \oplus b_3 \oplus b_4 \\ p_2 = b_1 \oplus b_3 \oplus b_4 \\ p_1 = b_1 \oplus b_2 \oplus b_4 \end{cases}$$

En el receptor se calcula un síndrome de 3 bits, utilizando las ecuaciones para el cálculo de los síndromes:

$$\begin{cases} s_3 = b_2 \oplus b_3 \oplus b_4 \oplus p_3 \\ s_2 = b_1 \oplus b_3 \oplus b_4 \oplus p_2 \\ s_1 = b_1 \oplus b_2 \oplus b_4 \oplus p_1 \end{cases}$$

Que indica si la palabra recibida es una palabra código válida:

$$\begin{cases} s_3 = 0 \\ s_2 = 0 \\ s_1 = 0 \end{cases}$$

O la posición, en binario, en la que se produjo el error, por ejemplo:

$$\begin{cases} s_3 = 0 \\ s_2 = 1 \\ s_1 = 1 \end{cases}$$

Indica que el error está en el bit que se encuentra en la posición 3 (b_1).

Para corregir el error, simplemente se cambia el bit en esa posición, si estaba en 0 por 1 o viceversa.

En la práctica, para obtener las ecuaciones para el cálculo de los bits de paridad y los bits del síndrome, se puede recurrir a una tabla como la Tabla 6.

Tabla 6. Obtención de las ecuaciones para el cálculo de los bits de paridad y de síndromes

	b_4	b_3	b_2	p_3	b_1	p_2	p_1
s_3	1	1	1	1	0	0	0
s_2	1	1	0	0	1	1	0
s_1	1	0	1	0	1	0	1

En la Tabla 6, en la fila de encabezado y columna de encabezado, se escriben los bits de la palabra código y del síndrome en forma genérica respectivamente. Cada columna de la tabla se completa, en números binarios, con la posición que ocupa cada uno de los bits en la palabra código.

Las ecuaciones para el cálculo de los bits de paridad se obtienen de la siguiente manera. Para obtener la ecuación para el cálculo del bit de paridad p_1, nos posicionamos en la tabla en la fila para el cual el bit de paridad p_1 está en 1 (en este caso, la fila 3). Sumando (módulo 2) los restantes bits de esa fila que estén en 1, se obtiene la ecuación para el cálculo del bit de paridad (Fig. 1):

$$p_1 = b_1 \oplus b_2 \oplus b_4$$

De manera similar, se obtienen las ecuaciones para el cálculo del bit de paridad p_2 y p_3.

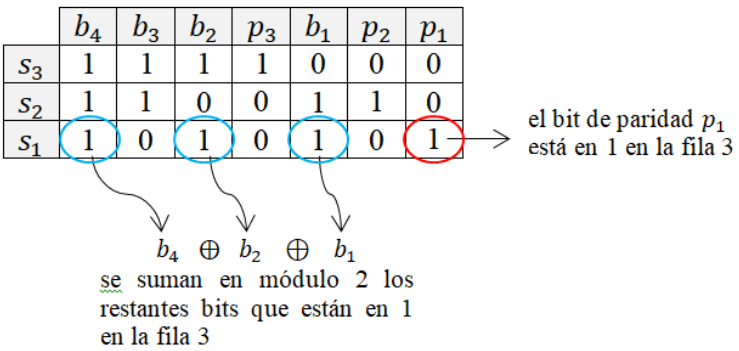

Fig. 1. Obtención de la ecuación para el cálculo del bit de paridad p_1

Las ecuaciones para el cálculo de los bits de síndrome se obtienen de la siguiente manera. Para obtener la ecuación para el cálculo del bit s_1, en la tabla se suman (modulo 2) todos los bits que estén en 1 para la fila s_1, con lo que se obtiene (Fig. 2):

$$s_1 = p_1 \oplus b_1 \oplus b_2 \oplus b_4$$

De manera similar, se obtienen las ecuaciones para el cálculo de los bits s_2 y s_3.

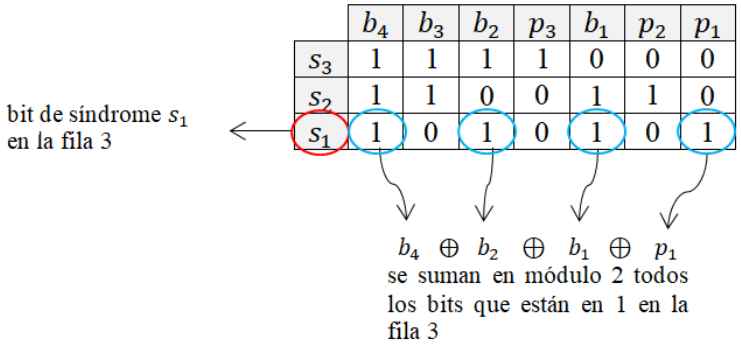

Fig. 2. Obtención de la ecuación para el cálculo del bit de síndrome s_1

Ejemplo 4. A partir de las ecuaciones de paridad y de síndrome descriptas anteriormente, codificar la palabra de datos $d_1 = 1110$ y decodificar la palabra código $c_1 = 1011100$, si se detecta un error, corregir indicando la posición del bit alterado.

Codificación: la palabra de datos es $d_1 = 1110$

Los bits de datos son:

b_4	b_3	b_2	b_1
1	1	1	0

Obtenemos los bits de paridad (utilizando las ecuaciones de paridad):

$$\begin{cases} p_3 = 1 \oplus 1 \oplus 1 = 1 \\ p_2 = 1 \oplus 1 \oplus 1 = 1 \\ p_1 = 0 \oplus 1 \oplus 1 = 0 \end{cases}$$

Formamos la palabra código:

b_4	b_3	b_2	$\boldsymbol{p_3}$	b_1	$\boldsymbol{p_2}$	$\boldsymbol{p_1}$
1	1	1	**1**	0	**1**	**0**

La palabra código es: $c_1 = 1111010$

Decodificación: la palabra código es $c_1 = 1011100$

Los bits de la palabra código son:

b_4	b_3	b_2	$\boldsymbol{p_3}$	b_1	$\boldsymbol{p_2}$	$\boldsymbol{p_1}$
1	0	1	**1**	1	**0**	**0**

Obtenemos los bits del síndrome:

$$\begin{cases} s_3 = 1 \oplus 0 \oplus 1 \oplus 1 = 1 \\ s_2 = 1 \oplus 0 \oplus 1 \oplus 0 = 0 \\ s_1 = 1 \oplus 1 \oplus 1 \oplus 0 = 1 \end{cases}$$

El síndrome es $S = 101$, la palabra recibida no es válida. El bit erróneo se encuentra en la posición 5 (b_2). Para corregir el error, se cambia el bit 1 por un bit 0

b_4	b_3	b_2	$\boldsymbol{p_3}$	b_1	$\boldsymbol{p_2}$	$\boldsymbol{p_1}$
1	0	**0**	**1**	1	**0**	**0**

La palabra código valida es $c_1 = 1001100$.

Para recuperar la palabra de datos, se descartan los bits de paridad, que se encuentran en posiciones potencia de dos y se obtiene:

$$d_1 = 1001$$

Un procedimiento similar se utiliza para construir otros códigos, por ejemplo el $(15{,}11{,}3)$ código de Hamming utiliza 4 bits de paridad para codificar palabras de datos de 11 bits.

6. Codificación de códigos de bloques lineales

Recordemos que un código de bloque lineal toma mensajes de k bits y los convierte en códigos de n bits mediante operaciones lineales sobre los bits del mensaje.

Si el código está en forma sistemática (Fig. 3), cada palabra código consiste de k bits del mensaje $d_1 d_2 d_3 \dots d_k$ seguido por $n - k$ bits de paridad $p_1 p_2 \dots p_{n-k}$, siendo p_i alguna combinación lineal de los d_i.

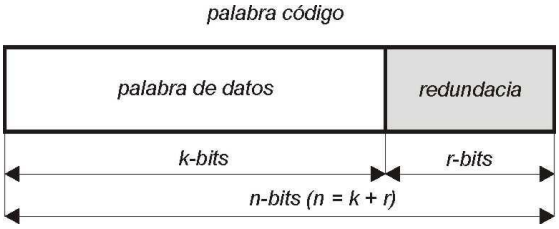

Fig. 3. Forma sistemática de un código de bloque lineal

Puesto que la transformación de palabra de datos a palabra código es lineal, se puede representar mediante notación matricial:

$$c = d * G$$

Dónde d es una matriz de $1 \times k$ elementos (un vector fila) con los bits de la palabra de datos; c es una matriz de $1 \times n$ elementos (vector fila) con los bits de la palabra código y G es una matriz de $k \times n$ elementos que caracteriza completamente al código de bloque lineal.

Si el código está en forma sistemática, la palabra código tiene la forma:

$$c = d_k \dots d_3 d_2 d_1 p_{n-k} \dots p_2 p_1$$

La matriz generadora G se descompone en una matriz identidad de $k \times k$ concatenada horizontalmente con una matriz P de $k \times (n - k)$ elementos:

$$G_{k\times n} = I_{k\times k} \mid P_{k\times(n-k)}$$

El procedimiento para codificar cualquier código de bloque lineal es sencillo: dada la matriz generadora G y una palabra de datos d con k bits, para generar la palabra código deseada se utiliza la ecuación:

$$c = d * G$$

7. Decodificación por síndrome

Es una forma eficiente para decodificar códigos de bloque lineales.

Introducimos una nueva H, llamada matriz de control de paridad, que es la concatenación horizontal de la traspuesta de la matriz P de la matriz G y una matriz identidad I de $(n-k)\times(n-k)$ elementos:

$$H = P^T \mid I_{(n-k)\times(n-k)}$$

Para cualquier palabra código c valida:

$$c * H^T = 0$$

Por lo tanto, para cualquier palabra código r recibida sin errores:

$$r * H^T = 0$$

Si una palabra r tiene errores, puede escribirse:

$$r = c \oplus e$$

Dónde c es una palabra código válida y e es un patrón de error, representado por un vector fila de $1 \times n$ elementos. Para una palabra r con errores, entonces:

$$r * H^T = (c \oplus e) * H^T = 0 \oplus e * H^T$$

Si r tiene al menos un error, entonces e estará compuesto por ceros y al menos un uno. En este caso, (para errores simples), habrá $n+1$ casos posibles de $e * H^T$: n se corresponden con exactamente un bit erróneo y 1 con el caso de bits sin errores, para el cual $e * H^T = 0$.

Estos $n+1$ posibles vectores, son los síndromes que representan diferentes patrones de error:

$$h(e) = e * H^T$$

Ejemplo 5. Sea el $(7,4,3)$ código con matriz generadora G y matriz de control de paridad H:

$$G = \begin{vmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 \end{vmatrix}; H = \begin{vmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{vmatrix}$$

1. Codificar la palabra de datos $d_1 = 011$.
2. Construir la tabla estándar reducida.
3. Decodificar las palabras recibidas $r_1 = 0110110$ y $r_2 = 0100110$

1. La palabra código se obtiene con $c = d * G$:

$$c_1 = d_1 * G = |0 \quad 1 \quad 1| * \begin{vmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 \end{vmatrix} = |0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 1 \quad 0|$$

La palabra condigo es $c_1 = 0110110$

2. La tabla estándar reducida (Tabla 7) se construye de la siguiente manera: en la columna líderes (e) se escriben los $n + 1$ patrones de error y a continuación se calcula el síndrome $h(e)$ para cada uno.

Tabla 7. Tabla estándar reducida

Líderes (e)	$h(e) = e * H^T$
0000000	0000
0000001	0001
0000010	0010
0000100	0100
0001000	1000
0010000	1011
0100000	1101
1000000	1110

3. La decodificación consiste en calcular el síndrome de la palabra recibida:

$$h(r) = r * H^T$$

$$si \begin{cases} h(r) = 0000, entonces\ r \in C\ (no\ hay\ error) \\ h(r) \neq 0000, entonces\ r \notin C\ (hay\ error) \end{cases}$$

Se busca en la tabla estándar el patrón de error o líder (e), asociado al síndrome obtenido que, sumado a la palabra r recibida, permite corregir el error y obtener la palabra código válida.

$$c = r \oplus e;\ c \in C$$

Para recuperar la palabra de datos, se descartan los $(n - k)$ bits de redundancia.

$$h(0110110) = |0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 1 \quad 0| * \begin{vmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} = |0 \quad 0 \quad 0 \quad 0|$$

$$h(r) = 0000, entonces\ r \in C\ (no\ hay\ error)$$

La palabra de datos es $d = 011$

$$h(0100110) = |0 \quad 1 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0| * \begin{vmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} = |1 \quad 0 \quad 1 \quad 1|$$

$$h(r) = 1011, entonces\ r \notin C\ (hay\ error)$$

Por tabla el síndrome 1011 tiene asociado el líder 0010000 que sumado a la palabra recibida 0100110, permite corregir el error y obtener la palabra código válida:

$$\begin{array}{cccccccc} & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ \oplus & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\ \hline & 0 & 1 & \mathbf{1} & 0 & 1 & 1 & 0 \end{array}$$

La palabra de datos es $d = 011$

Bibliografía recomendada

[1] D. L. La Red Martínez. Presentaciones de Clases Teóricas. Comunicaciones de Datos, Facultad de Ciencias Exactas y Naturales y Agrimensura. Universidad Nacional del Nordeste.

[2] A. S. Tanenbaum. *Redes de Computadoras*, 4ta. Edición, PEARSON Educación, México, 2003.

[3] W. Stallings. *Comunicaciones y Redes de Computadoras*, 6ta. Edición. Prentice Hall, Madrid, 2000.