UNIVERSIDAD DEL VALLE DE GUATEMALA
CC3067 Redes
Sección 10
Ing. Jorge Yass

Laboratorio 2 - Parte 1
Esquemas de detección y corrección de errores

SEBASTIAN ARISTONDO PEREZ 20880
JOSE DANIEL GONZALEZ CARRILLO 20293

GUATEMALA, 10 de septiembre de 2018

Descripción de la práctica

Durante la presente práctica de laboratorio se llevó a cabo la implementación de dos algoritmos relacionados a los errores en mensajes. El primero fue CRC-32 y el segundo fue el código de Hamming. Para ambos algoritmos se debía implementar un emisor y un receptor. El emisor se encargaba de generar una trama, que era una secuencia de bits (1s y 0s) de cualquier longitud. Para esto, cada emisor debía hacer cálculos respectivos para añadir ciertos bits de referencia, que sirvieron para identificar errores en la trama. Por otra parte, los receptores permitieron el ingreso de la trama generada y posteriormente realizaron operaciones para detectar errores y corregir estos en el caso de Hamming.

Tanto el emisor como el receptor debían estar programados en diferentes lenguajes. Nosotros elegimos programar los dos emisores en Java y los dos receptores en Python. Adicionalmente, al momento de ingresar las tramas generadas por los emisores, nosotros debimos modificar manualmente ciertos bits para inducir errores. Se probaron tres tramas distintas sin modificar, tres tramas con un bit modificado, tres tramas con dos bits modificados y una trama modificada especialmente para que el algoritmo no pudiera detectarlo.

El CRC-32 (Cyclic Redundancy Check 32-bit) es un algoritmo de detección de errores ampliamente utilizado en comunicaciones y almacenamiento de datos. Este se basa en la configuración de un polinomio generador, el cual al ser un algoritmo estandarizado siempre se utiliza el mismo. Este polinomio es representado de forma binaria siendo 1 los coeficientes de las potencias que si se encuentran en el polinomio y 0 los coeficientes que no. Para codificar el mensaje entrante se le agregan 32 ceros al final, luego con el polinomio de CRC-32 se comienza a realizar una “división” en la cual se utiliza la función XOR para poder operar el dividendo (el mensaje aplicado con ceros) y el divisor (el polinomio). Una vez se obtiene el residuo de esta división, ese residuo se agrega en vez de los ceros colocados anteriormente, únicamente sustituyendo la cantidad de ceros según el tamaño del residuo. Para verificar si el mensaje podría ser íntegro se realiza la misma operación buscando que el resultado de la división sea 0 en ese caso podemos decir que el mensaje podría ser íntegro en caso contrario lo rechazamos porque se detectaron errores.(Mouse, 2019)

El código de Hamming es un tipo de código de corrección de errores utilizado para detectar y corregir errores en la transmisión y almacenamiento de datos. La principal característica del Código de Hamming es que agrega bits de paridad adicionales a los datos originales para detectar y corregir errores. Estos bits de paridad se calculan utilizando reglas específicas y se colocan en ciertas posiciones dentro de la secuencia de datos. Para codificar los datos originales, se determina la cantidad de bits de paridad necesarios para cubrir todos los bits de información. Se eligen las posiciones en la secuencia de datos donde se ubicarán estos bits de paridad. Los bits de paridad se calculan utilizando reglas matemáticas específicas. Por ejemplo, en un código de Hamming de un solo error, los bits de paridad se colocan en posiciones que son potencias de dos (1, 2, 4, 8, etc.). Cada bit de paridad se calculará en función de ciertos bits de información. Una vez calculados, los bits

de paridad se insertan en las posiciones adecuadas dentro de la secuencia de datos original, aumentando su longitud total. (Arboix, 2012)

Durante la transmisión o el almacenamiento de datos, si ocurren errores, los bits de paridad permiten detectar dichos errores. Si se altera un solo bit, se produce una discrepancia entre los bits de paridad calculados y los recibidos. Dependiendo del tipo de código de Hamming utilizado (por ejemplo, Hamming(7,4) o Hamming(8,4)), el código puede corregir errores de un solo bit o solo detectar errores. Si se detecta un error, es posible determinar qué bit está incorrecto y corregirlo en base a los bits de paridad.(Arboix, 2012)

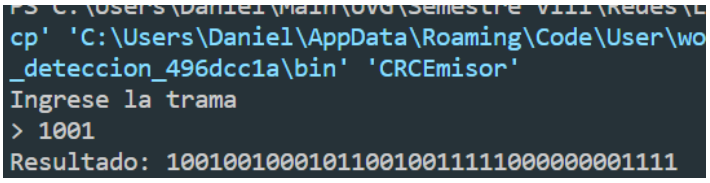
Resultados

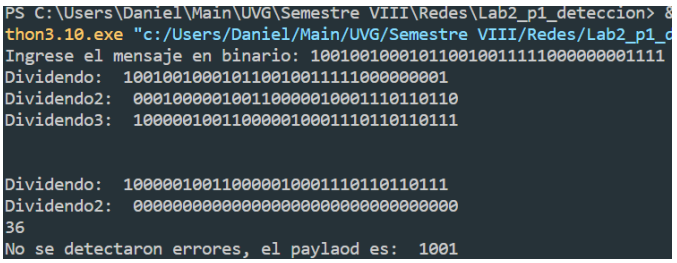
Tramas utilizadas

Correctas

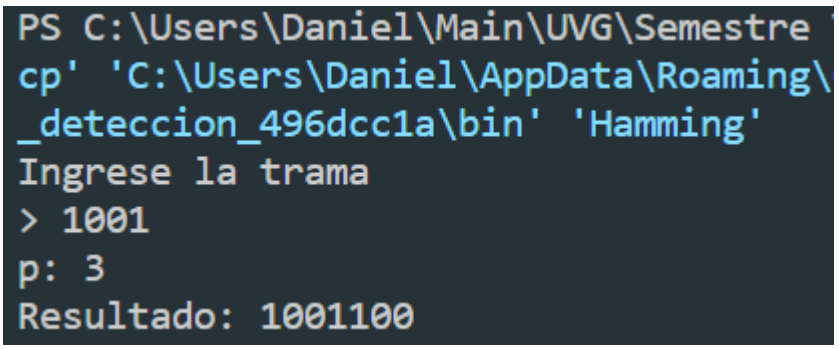
1001

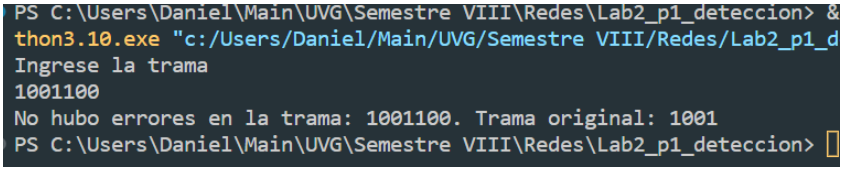
CRC-32





Hamming





110101

CRC-32

```
cp' 'C:\Users\Daniel\AppData\Roaming\Code\User\workspac
_deteccion_496dcc1a\bin' 'CRCEmisor'
Ingrese la trama
> 110101
Resultado: 11010111000011111101110000011011111011
```

```
PS C:\Users\Daniel\Main\UVG\Semestre VIII\Redes\Labz
thon3.10.exe "c:/Users/Daniel/Main/UVG/Semestre VIII/
Ingrese el mensaje en binario: 11010111000011111101111
Dividendo:  11010111000011111101110000110111
Dividendo2:  01010101011011110101001011000000
Dividendo3:  10101010110111101010010110000001


Dividendo:  10101010110111101010010110000001
Dividendo2:  00101000101111100010101101010110
Dividendo3:  10100010111110001010110101101010


Dividendo:  10100010111110001010110101101010
Dividendo2:  00100000100110000010001101101101
Dividendo3:  10000010011000001000111010110111


Dividendo:  10000010011000001000111010110111
Dividendo2:  00000000000000000000000000000000
38
No se detectaron errores, el paylaod es:  110101
```

Hamming

```
Ingrese la trama
> 110101
p: 4
Resultado: 1100101110
```

```
Ingrese la trama
1100101110
No hubo errores en la trama: 1100101110. Trama original: 110101
```

11100101001

CRC-32

```
cp' 'C:\Users\Daniel\AppData\Roaming\Code\User\workspaceSt
_deteccion_496dcc1a\bin' 'CRCEmisor'
Ingrese la trama
> 11100101001
Resultado: 11100101001100011010010010100111000100101010
```

```
thon3.10.exe "c:/Users/Daniel/Main/UVG/Semestre VIII/Redes/Lab2_
Ingrese el mensaje en binario: 11100101001100011010010010100111
Dividendo:  11100101001100011010010010100111
Dividendo2:  01100111010100010010100111001
Dividendo3:  11001110101000100101001111110010


Dividendo:  11001110101000100101001111110010
Dividendo2:  01001100110000101110101000100101
Dividendo3:  10011001100001011011010001001010


Dividendo:  10011001100001011011010001001010
Dividendo2:  00011011111001010011101010111101
Dividendo3:  11011111001010011101010111101100


Dividendo:  11011111001010011101010111101100
Dividendo2:  01011101010010010111010001011011
Dividendo3:  10111010100100101011010010110111


Dividendo:  10111010100100101011010010110111
Dividendo2:  00111000111100100011101010000000
Dividendo3:  11100011110010001110101000000001


Dividendo:  11100011110010001110101000000001
Dividendo2:  01100001101010000110010011010110
Dividendo3:  11000011010100001100100110110100


Dividendo:  11000011010100001100100110110100
Dividendo2:  01000001001100000100011101101011
Dividendo3:  10000010011000001000111011010111


Dividendo:  10000010011000001000111011010111
Dividendo2:  00000000000000000000000000000000
43
No se detectaron errores, el paylaod es:  11100101001
```

Hamming

```
> 11100101001
p: 4
Resultado: 111001001000110
```

```
Ingrese la trama
111001001000110
No hubo errores en la trama: 111001001000110. Trama original: 11100101001
```

Modificadas por 1 bit

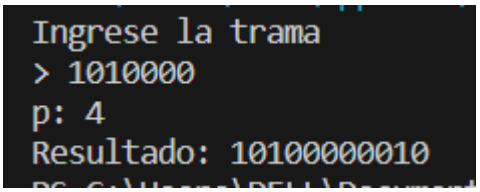
CRC-32
Original: 1010 -> 10100010111110001010110101101101011010110 ; Modificada: 10100010111110001010110101101101**1**1010

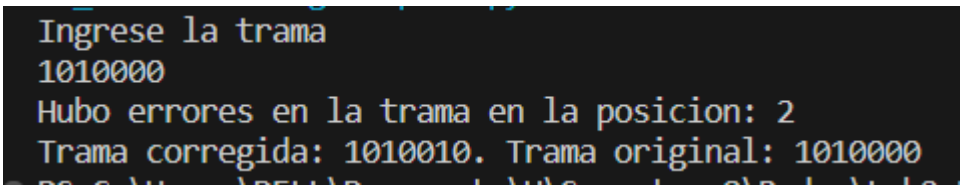
```
cp' 'C:\Users\Daniel\AppData\Roaming\Code\User\works
_deteccion_496dcc1a\bin' 'CRCEmisor'
Ingrese la trama
> 1010
Resultado: 10100010111110001010110101101101011010110
```

```
thon3.10.exe "c:/Users/Daniel/Main/UVG/Semestre VIII/Redes/Lab2_p1_de
Ingrese el mensaje en binario: 10100010111110001010110101101101011010110
Dividendo:  10100010111110001010110101101101011010110
Dividendo2:  00100000100110000010001110110110011
Dividendo3:  10000010011000001000111011011001l


Dividendo:  10000010011000001000111011011001l
Dividendo2:  00000000000000000000000000000100
Dividendo3:  1000


36
Se detectaron errores, por lo tanto la trama se descarta
```

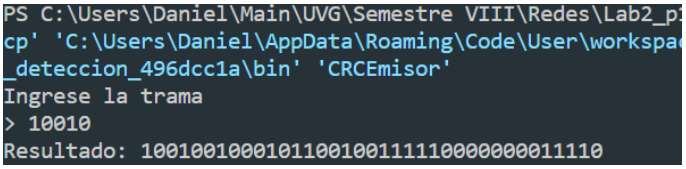
Hamming

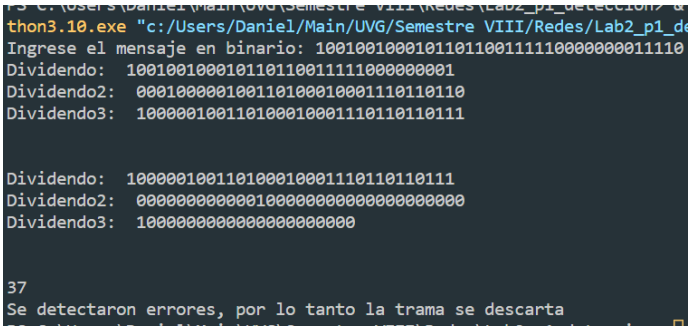
Original: 1010 -> 1010010 ; Modificada: 101000



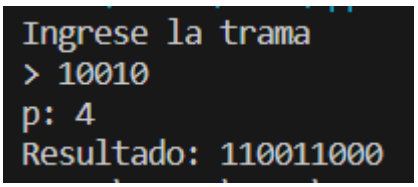


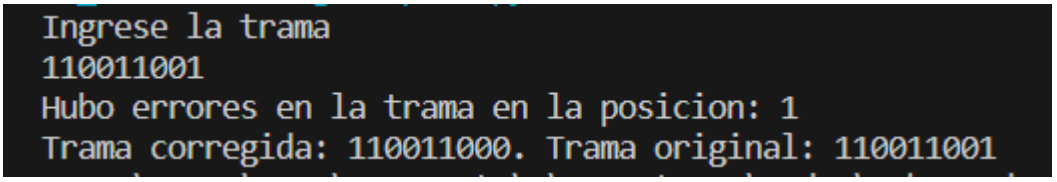
CRC-32
Original: 10010 -> 10010010001011001001111100000000011110; Modificada: 10010010001011011001111100000000011110





Hamming
Original: 10010 -> 110011000; Modificada: 110011001





CRC-32
Original: 11000 -> 11000011010100001100100110110110010000; Modificada: 10000011010100001100100110110110010000

```
PS C:\Users\Daniel\Main\UVG\Semestre VIII\Redes\Lab2_
cp' 'C:\Users\Daniel\AppData\Roaming\Code\User\worksp
_deteccion_496dcc1a\bin' 'CRCEmisor'
Ingrese la trama
> 11000
Resultado: 1100001101010000110010011011011001000
```

```
PS C:\Users\Daniel\Main\UVG\Semestre VIII\Redes\Lab2_p1_deteccion> &
thon3.10.exe "c:/Users/Daniel/Main/UVG/Semestre VIII/Redes/Lab2_p1_de
Ingrese el mensaje en binario: 1000001101010000110010011011011001000
Dividendo:  1000001101010000110010011011011001000
Dividendo2:  0000000100110000010000111011011011
Dividendo3:  1001100000100011101101101110000


37
Se detectaron errores, por lo tanto la trama se descarta
```

Hamming

Original: 11000 -> 111001010 ; Modificada: 111101010

```
> 11000
p: 4
Resultado: 111001010
```

```
Ingrese la trama
111101010
Hubo errores en la trama en la posicion: 6
Trama corregida: 111001010. Trama original: 111101010
```

Modificadas por 2 bits

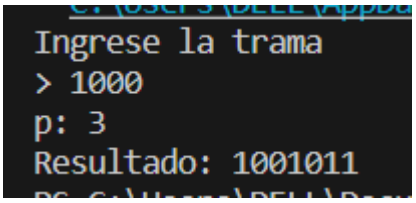
CRC-32

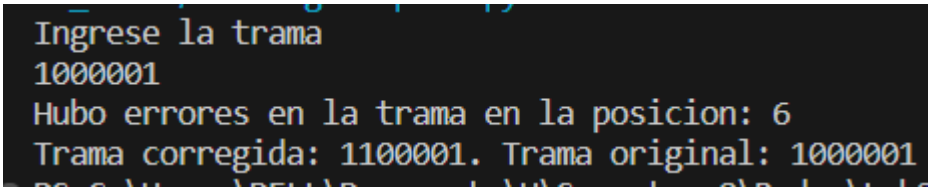
Original: 1000 -> 1000001001100000100011101101101110000; Modificada: 1000001000100000100011111101101110000

```
PS C:\Users\Daniel\Main\UVG\Semestre VIII\Redes\L
cp' 'C:\Users\Daniel\AppData\Roaming\Code\User\wo
_deteccion_496dcc1a\bin' 'CRCEmisor'
Ingrese la trama
> 1000
Resultado: 1000001001100000100011101101101110000
```

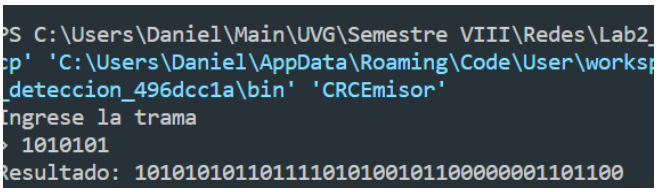
```
PS C:\Users\Daniel\Main\UVG\Semestre VIII\Redes\Lab2_p1_deteccion> &
thon3.10.exe "c:/Users/Daniel/Main/UVG/Semestre VIII/Redes/Lab2_p1_de
Ingrese el mensaje en binario: 1000001000100000100011111101101110000
Dividendo:  1000001000100000100011111101101111
Dividendo2:  0000000001000000000000001000000000
Dividendo3:  1000000000000001000000000000


36
Se detectaron errores, por lo tanto la trama se descarta
```

Hamming
Original: 1000 -> 1001011; Modificada: 1000001

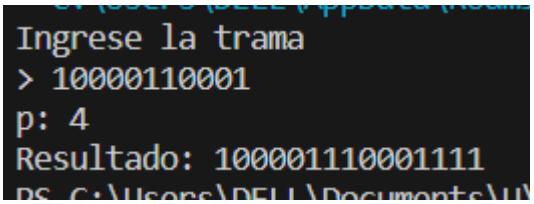


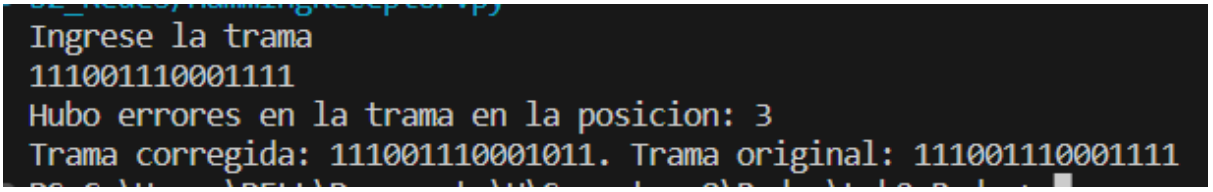


CRC-32
Original: 1010101 -> 10101010110111101010010110000000110110 0; Modificada: 10101010100111101010010110000000110110 1



Hamming
Original: 10000110001 ->100001110001111; Modificada: 1 1100111000111 1





CRC-32
Original: 10000110001 -> 10000110001100101011101011101010000110001110; Modificada: 10100110001100111011101011101010000110001110

```
cp' 'C:\Users\Daniel\AppData\Roaming\Code\User\workspaceSt
_deteccion_496dcc1a\bin' 'CRCEmisor'
Ingrese la trama
> 10000110001
Resultado: 10000110001100101011101011101010000110001110
```

```
thon3.10.exe "c:/Users/Daniel/Main/UVG/Semestre VIII/Redes/Lab2_pi_deteccio
Ingrese el mensaje en binario: 10100110001100111011101011101010000110001110
Dividendo:  10100110001100111011101011101010
Dividendo2:  00100100010100110011010000100011
Dividendo3:  10010001010011001101000011000101


Dividendo:  10010001010011001101000011000101
Dividendo2:  00010011001011000101111000011010
Dividendo3:  10011001011000101111000011010100


Dividendo:  10011001011000101111000011010100
Dividendo2:  00011011000000100111111000100011
Dividendo3:  11011000000100111111000100011011


Dividendo:  11011000000100111111000100011011
Dividendo2:  01011010011100110111111010101100
Dividendo3:  10110100111001101111111010011001


Dividendo:  10110100111001101111111010011001
Dividendo2:  00110110100001100111000001110110
Dividendo3:  11011010000110011100000111011100


43
Se detectaron errores, por lo tanto la trama se descarta
```

Hamming
Original: 10000110001 -> 100001110001111 ; Modificada: 000001110001110

```
Ingrese la trama
> 10000110001
p: 4
Resultado: 100001110001111
```

```
Ingrese la trama
000001110001110
Hubo errores en la trama en la posicion: 14
Trama corregida: 010001110001110. Trama original: 000001110001110
```

Especialmente modificadas para que errores no puedan ser detectados

- CRC-32: Se usó 1000
    - Original: 10000010011000001000111011011011000
    - Modificada: 10010010001011001001111100000001111

```
Ingrese la trama
> 1000
Resultado: 10000010011000001000111011011011000
```

```
Ingrese el mensaje en binario: 10010010001011001001111100000001111
Dividendo:  10010010001011001001111100000001
Dividendo2:  00010000010011000001000111011011O
Dividendo3:  10000010011000001000111011011O111


Dividendo:  10000010011000001000111011011O111
Dividendo2:  00000000000000000000000000000000
36
No se detectaron errores, el paylaod es:  1001
```

- Hamming: Se usó 1000
    - Original: 1001011
    - Modificada: 1001100

```
Ingrese la trama
> 1000
p: 3
Resultado: 1001011
```

```
Ingrese la trama
1001100
No hubo errores en la trama: 1001100. Trama original: 1001
```

Discusión

El CRC-32 (Cyclic Redundancy Check 32-bit) es un algoritmo de detección de errores ampliamente utilizado en comunicaciones y almacenamiento de datos. Gracias a que es un algoritmo estandarizado el polinomio que se utiliza es el resultado de múltiples pruebas e iteraciones. Consiguiendo así el mejor resultado en cuanto a la codificación de las tramas. A pesar de que existe la posibilidad de que los bits sufran flip y llegue otro mensaje el cual no presente errores como se demostró en la práctica, para que esta situación se de deben ocurrir flips muy específicos. Se buscó encontrar algún patrón, cambiar bits al azar para ver si se conseguía que el algoritmo no encontrase errores. Pero tras muchos intentos fallidos vimos que la forma más sencilla de lograr que la trama pasará desapercibida era transformar una trama válida A en una trama válida B.

La trama A será el resultado de codificar el mensaje 1000 por lo tanto A sería 10000010011000001000111011011O111000, la trama B sería el resultado de codificar el mensaje 1001 por lo tanto B sería 10010010001011001001111100000001111. Por lo tanto la interferencia o ruido debería de flipear los bits de la siguiente forma 100**1**0010**0**10**11**0010**0**1111**100**0**00**0**0**1**111** para lograr pasar como un mensaje

posiblemente íntegro. Lo que implica que la transformación que debe sufrir el mensaje es tomar la forma de otra trama codificada correctamente

Durante las pruebas con errores para el código de Hamming fue posible observar que cuando solo se modificó un bit, el algoritmo fue bastante efectivo para detectar el error y modificarlo. Sin embargo, al momento de modificar 2 bits, el algoritmo si pudo detectar un error, pero no fue exacto para detectar su posición. Cuando se tienen más de 2 bits modificados y se realiza el proceso de revisar las posiciones correspondientes de la trama para los bits de paridad por medio de la tabla de verdad, es posible que se encuentren múltiples conjuntos que no tienen una cantidad par de 1s. Entonces lo que sucede es que al momento de convertir este resultado de base 2 a base 10, se obtendrá un número que no corresponde a las posiciones donde se encuentran los bits erróneos. Sumado a esto, el algoritmo sólo sabe corregir un error, por eso el resultado que se pasa de base 2 a base 10.

Por otra parte, generar una trama que tuviera error y fuera indetectable para el algoritmo fue una tarea relativamente sencilla. Se lograron idear dos métodos para hacerlo. El primero fue identificar las posiciones correspondientes con 1s en la tabla de verdad para cada bit de paridad. Luego se evaluó si estas posiciones en conjunto tenían una cantidad de bits par. Si no lo tenían, se modificó alguno de los bits de forma que la cantidad fuera par. Esto se repitió para todas las posiciones con 1s de la tabla de verdad con cada bit de paridad, cuidando que todos dieran una cantidad par. De esta forma, el algoritmo no identifica errores, porque no detecta 1s impares.

Comentarios

Esta práctica ayudó a afianzar los conocimientos respecto a los esquemas de detección y corrección de errores. Entender cómo funcionan estos algoritmos no es comparable con las dificultades que conlleva implementarlo desde cero. Ambos algoritmos fueron muy interesantes. En nuestra opinión, CRC-32 fue un algoritmo más sencillo en cuanto a implementación, porque tiene menos pasos diferentes que hacer. Sin embargo, la ventaja de la corrección de errores de Hamming puede permitir tener mejor performance en redes, porque no se necesitará volver a mandar los mensajes del emisor para que estén bien. Finalmente, podemos decir que la práctica fue bastante interesante y retadora, porque era necesario idear la forma de implementar los algoritmos.

Conclusiones

- Cuando se tiene una trama que tiene más de un bit erróneo, el algoritmo de Hamming no es eficaz para corregir los errores, pero si puede detectarlos.
- Como se puede observar en la complejidad de la modificación de trama para CRC-32, la posibilidad de que un error pase inadvertido es baja.
- La complejidad de calcular p en Hamming es directamente proporcional al tamaño de la trama

Citas y Referencias

Mouse, J. (2019, February 5). *CRC32: Verificación de Redundancia Cíclica*.

Www.jc-Mouse.net.

https://www.jc-mouse.net/java/crc32-verificacion-de-redundancia-ciclica

Invarato, R. (2016, October 5). *Código de Hamming: Detección y Corrección de errores*.

Jarroba. https://jarroba.com/codigo-de-hamming-deteccion-y-correccion-de-errores/