Universidad del Valle de Guatemala
Facultad de Ingeniería
Departamento de Ciencias de la Computación
Redes CC3067
Ing. Jorge Yass

Laboratorio # 2
Esquemas de detección y corrección de errores

Christopher García 20541
Maria Isabel Solano Bonilla 20504
Semestre II
Agosto 2023

Corrección de errores

La corrección de errores fue implementada con el código de Hamming. El emisor se encuentra programado en Java, y el receptor en Python. Ambos programas se comunican por medio de un archivo de texto escrito por Java y leído por Python, el cual contiene la cadena de bytes codificados. Para el manejo de bits se trabajaron con cadenas de strings y arreglos de ints de 0's y 1's que facilitaron la implementación del código.

En este código se busco cumplir la siguiente condición: (p + m + 1) <= 2^p.

Al iniciar el programa, del lado del emisor, se solicita al usuario la cadena a evaluar. Se calcula el tamaño de esta, se crea un diccionario para manejar las posiciones de mejor forma, ordenando la cadena original en todas las posiciones diferentes de una potencia de 2 (estas potencias se mencionan más adelante) de izquierda a derecha y posteriormente se inicia el código *per se*.

Siguiendo los pasos vistos en clase se procede a calcular p, siendo este el valor mínimo de todos los p posibles. Luego de obtener p se obtienen las potencias de 2 desde 2^0 hasta 2^p-1, con estas potencias se procedió a crear la tabla que almacenaría los bits en formas intermitentes dependiendo de la potencia que representan (i.e. potencia 2^2 = 4 entonces los bits irían, siempre comenzando desde 0, de la siguiente forma: 0,0,0,0,1,1,1,1,...) con esta tabla era posible determinar aquellos conjuntos de posiciones que se debían evaluar para cada potencia en la cadena original y determinar los bits de paridad necesarios para completar la respuesta final. Una vez creados estos conjuntos y determinados los bits de paridad se determinaron los bits necesarios para completar la cadena original y se agregaron al diccionario mencionado al principio, posteriormente se retorna la cadena completa como parte de la respuesta del emisor.

Del lado del receptor se recibe una cadena (modificada en el sentido de haber sido procesada por el emisor), y se realiza un proceso similar al del emisor ya que se calculan nuevamente las potencias, la tabla y demás cualidades para poder determinar si el mensaje cuenta con errores o no. En caso de que no cuente con errores, el conjunto de bits de paridad serán únicamente 0, o bien deberán coincidir con el conjunto de bits de paridad generados por el emisor. Si en algún caso fallara, el bit incorrecto será determinado por el número decimal proveniente de convertir el conjunto de bits de paridad de binario a decimal. Los resultados finales se presentarían de la forma solicitada en el laboratorio.

- Se imprime la data ingresada por el usuario
- Se imprime la data enviada por el emisor
- Se imprimen mensajes dependiendo del análisis
    - Si todo está bien, se muestra el mensaje y la trama
    - Si hay errores
        - Se descarta la trama
        - Se corrige el error y se muestra la trama corregida

Detección de errores

La detección de errores fue implementada con el algoritmo CRC-32. El emisor se encuentra programado en Java, y el receptor se encuentra programado en Python. Ambos programas se comunican por medio de un archivo de texto escrito por Java y leído por Python, el cual contiene la cadena de bytes codificados. A su vez, ambos programas, en lugar de manejar cadenas de caracteres o de ints de 0 y 1, se decidió que estos manejaran arreglos dinámicos o listas de booleanos verdadero y falso, significando 1 y 0 respectivamente.

Para el algoritmo de CRC-32 es necesario definir un polinomio de grado 32 - 1. El polinomio definido es el siguiente:
$x^{31} + x^{29} + x^{28} + x^{26} + x^{24} + x^{20} + x^{18} + x^{16} + x^{11} + x^{10} + x^{9} + x^{7} + x^{4} + x^{3} + x^{1}$
El cual en la práctica se traduce a la siguiente secuencia:
10110101000101010000111010011010

El inicio del programa le pide al usuario que ingrese una secuencia de 0 y 1 de cualquier tamaño. El programa hace las validaciones necesarias para asegurarse que la secuencia ingresada es una secuencia válida y con la que se puede proseguir para realizar el algoritmo. En caso contrario, se le requerirá al usuario ingresar otra cadena hasta que escriba una válida. Con la cadena validada, se le pasa al emisor, quien primero la convierte en la secuencia de verdaderos y falsos correspondientes. Con ello, y con el polinomio definido, ya se puede iniciar el algoritmo.

Para iniciar el algoritmo, se hace un XOR entre la trama y el polinomio. Esta cadena de largo 32 es almacenada en una variable llamada actual, la cual es de mucha utilidad para todo el algoritmo. También se crea una variable nextB la cual almacena la posición del último valor tomado de la trama. Luego de este paso, se procede a iniciar con un ciclo while, el cual corre hasta que nextB sea mayor a lo largo de la trama.

Durante el ciclo while se hace agrega el nuevo byte a la trama y se elimina el residuo 0 que siempre queda al inicio. Si el inicio de esta nueva cadena no es 0 se procede a realizar nuevamente la operación de or exclusivo. Este resultado se almacena en la variable actual, y así se procede con el ciclo while hasta que la condición se deja de cumplir. Finalmente el residuo final se agrega a la cadena original. Y ese es el mensaje codificado.

El mensaje codificado es transferido a un String, el cual luego se almacena en un archivo .txt. El archivo luego es leído por el Receptor, escrito en Python, el cual a partir de los 0 y 1 recrea la trama en un array de trues y falses. El receptor hace el exacto mismo proceso que el emisor, y de último verifica que la trama que ha quedado como residuo sean solo ceros.

Resultados

Corrección de errores:

Usando la trama 1001 se obtuvieron los siguientes resultados:
Emisor Java:

```
Data enviada por el emisor: 1001
Respuesta del emisor: 1001100
```

Receptor Python:

```
+---+----+----+----+
| n | P0 | P1 | P2 |
+---+----+----+----+
| 0 | 0  | 0  | 0  |
| 1 | 1  | 0  | 0  |
| 2 | 0  | 1  | 0  |
| 3 | 1  | 1  | 0  |
| 4 | 0  | 0  | 1  |
| 5 | 1  | 0  | 1  |
| 6 | 0  | 1  | 1  |
| 7 | 1  | 1  | 1  |
+---+----+----+----+

Data enviada al receptor:  1001100
Resultado: Todo correcto, el mensaje no cuenta con errores
```

Trama modificada

```python
newL = list(req)
newL[2] = '1'
bad_request = "".join(newL)
Rec = Receptor(bad_request, bitsA)
res,correcion = Rec.check()

print()
print("Data enviada al receptor: ", bad_request)
print("Resultado:", res)
if correcion != "":
    print(correcion)
print()
```

Corrección de errores

```
+---+----+----+----+
| n | P0 | P1 | P2 |
+---+----+----+----+
| 0 | 0  | 0  | 0  |
| 1 | 1  | 0  | 0  |
| 2 | 0  | 1  | 0  |
| 3 | 1  | 1  | 0  |
| 4 | 0  | 0  | 1  |
| 5 | 1  | 0  | 1  |
| 6 | 0  | 1  | 1  |
| 7 | 1  | 1  | 1  |
+---+----+----+----+

Data enviada al receptor:  1011100
Resultado: Hubo un error en el bit 5
Se hizo la correcion. Trama final: 1001100
```

Usando la trama 11101001010101010011010101101001, se consiguieron los siguientes resultados:

Emisor Java

```
Data enviada por el emisor: 11101001010101010011010101101001
Respuesta del emisor: 11101000101010101001101101011001001101
```

Receptor Python

n	P0	P1	P2	P3	P4	P5
0	0	0	0	0	0	0
1	1	0	0	0	0	0
2	0	1	0	0	0	0
3	1	1	0	0	0	0
4	0	0	1	0	0	0
5	1	0	1	0	0	0
6	0	1	1	0	0	0
7	1	1	1	0	0	0
8	0	0	0	1	0	0
9	1	0	0	1	0	0
10	0	1	0	1	0	0
11	1	1	0	1	0	0
12	0	0	1	1	0	0
13	1	0	1	1	0	0
14	0	1	1	1	0	0
15	1	1	1	1	0	0
16	0	0	0	0	1	0
17	1	0	0	0	1	0
18	0	1	0	0	1	0
19	1	1	0	0	1	0
20	0	0	1	0	1	0
21	1	0	1	0	1	0
22	0	1	1	0	1	0
23	1	1	1	0	1	0
24	0	0	0	1	1	0
25	1	0	0	1	1	0
26	0	1	0	1	1	0
27	1	1	0	1	1	0
28	0	0	1	1	1	0
29	1	0	1	1	1	0
30	0	1	1	1	1	0
31	1	1	1	1	1	0
32	0	0	0	0	0	1
33	1	0	0	0	0	1

33	1	0	0	0	0	1
34	0	1	0	0	0	1
35	1	1	0	0	0	1
36	0	0	1	0	0	1
37	1	0	1	0	0	1
38	0	1	1	0	0	1
39	1	1	1	0	0	1
40	0	0	0	1	0	1
41	1	0	0	1	0	1

```
Data enviada al receptor:  11101000101010101001101101011001001101
Resultado: Todo correcto, el mensaje no cuenta con errores
```

Con la trama 11010010100101010010101010110100110 obtenemos:

```
Código de Hamming

Data enviada por el emisor: 11010010100101010010101010110100110
Respuesta del emisor: 11010010100110101001010101011011010110001

Archivo creado con éxito
```

n	P0	P1	P2	P3	P4	P5
0	0	0	0	0	0	0
1	1	0	0	0	0	0
2	0	1	0	0	0	0
3	1	1	0	0	0	0
4	0	0	1	0	0	0
5	1	0	1	0	0	0
6	0	1	1	0	0	0
7	1	1	1	0	0	0
8	0	0	0	1	0	0
9	1	0	0	1	0	0
10	0	1	0	1	0	0
11	1	1	0	1	0	0
12	0	0	1	1	0	0
13	1	0	1	1	0	0
14	0	1	1	1	0	0
15	1	1	1	1	0	0
16	0	0	0	0	1	0
17	1	0	0	0	1	0
18	0	1	0	0	1	0
19	1	1	0	0	1	0
20	0	0	1	0	1	0
21	1	0	1	0	1	0
22	0	1	1	0	1	0
23	1	1	1	0	1	0
24	0	0	0	1	1	0
25	1	0	0	1	1	0

25	1	0	0	1	1	0
26	0	1	0	1	1	0
27	1	1	0	1	1	0
28	0	0	1	1	1	0
29	1	0	1	1	1	0
30	0	1	1	1	1	0
31	1	1	1	1	1	0
32	0	0	0	0	0	1
33	1	0	0	0	0	1
34	0	1	0	0	0	1
35	1	1	0	0	0	1
36	0	0	1	0	0	1
37	1	0	1	0	0	1
38	0	1	1	0	0	1
39	1	1	1	0	0	1
40	0	0	0	1	0	1
41	1	0	0	1	0	1
42	0	1	0	1	0	1
43	1	1	0	1	0	1
44	0	0	1	1	0	1
45	1	0	1	1	0	1
46	0	1	1	1	0	1

```
Data enviada al receptor:  11010010100111010100101010101101101010110001
Resultado: Todo correcto, el mensaje no cuenta con errores
```

Con la trama 1010010100100

```
Código de Hamming

Data enviada por el emisor: 1010010100100
Respuesta del emisor: 101100101010101001

Archivo creado con éxito
```

n	P0	P1	P2	P3	P4
0	0	0	0	0	0
1	1	0	0	0	0
2	0	1	0	0	0
3	1	1	0	0	0
4	0	0	1	0	0
5	1	0	1	0	0
6	0	1	1	0	0
7	1	1	1	0	0
8	0	0	0	1	0
9	1	0	0	1	0
10	0	1	0	1	0
11	1	1	0	1	0
12	0	0	1	1	0
13	1	0	1	1	0
14	0	1	1	1	0
15	1	1	1	1	0
16	0	0	0	0	1
17	1	0	0	0	1
18	0	1	0	0	1
19	1	1	0	0	1
20	0	0	1	0	1

```
Data enviada al receptor:  101100101010101001
Resultado: Todo correcto, el mensaje no cuenta con errores
```

Con la trama: 110101

```
Código de Hamming

Data enviada por el emisor: 110101
Respuesta del emisor: 1100101110

Archivo creado con éxito
```

n	P0	P1	P2	P3
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	1	1	0	0
4	0	0	1	0
5	1	0	1	0
6	0	1	1	0
7	1	1	1	0
8	0	0	0	1
9	1	0	0	1
10	0	1	0	1
11	1	1	0	1

```
Data enviada al receptor:  1100101110
Resultado: Todo correcto, el mensaje no cuenta con errores
```

```
+----+----+----+----+----+
| n  | P0 | P1 | P2 | P3 |
+----+----+----+----+----+
| 0  | 0  | 0  | 0  | 0  |
| 1  | 1  | 0  | 0  | 0  |
| 2  | 0  | 1  | 0  | 0  |
| 3  | 1  | 1  | 0  | 0  |
| 4  | 0  | 0  | 1  | 0  |
| 5  | 1  | 0  | 1  | 0  |
| 6  | 0  | 1  | 1  | 0  |
| 7  | 1  | 1  | 1  | 0  |
| 8  | 0  | 0  | 0  | 1  |
| 9  | 1  | 0  | 0  | 1  |
| 10 | 0  | 1  | 0  | 1  |
| 11 | 1  | 1  | 0  | 1  |
+----+----+----+----+----+

Data enviada al receptor:  1100101110
Data modificada enviada al receptor:  1110101110

Resultado: Hubo un error en el bit 3
Trama descartada por errores
```

```
+----+----+----+----+----+
| n  | P0 | P1 | P2 | P3 |
+----+----+----+----+----+
| 0  | 0  | 0  | 0  | 0  |
| 1  | 1  | 0  | 0  | 0  |
| 2  | 0  | 1  | 0  | 0  |
| 3  | 1  | 1  | 0  | 0  |
| 4  | 0  | 0  | 1  | 0  |
| 5  | 1  | 0  | 1  | 0  |
| 6  | 0  | 1  | 1  | 0  |
| 7  | 1  | 1  | 1  | 0  |
| 8  | 0  | 0  | 0  | 1  |
| 9  | 1  | 0  | 0  | 1  |
| 10 | 0  | 1  | 0  | 1  |
| 11 | 1  | 1  | 0  | 1  |
+----+----+----+----+----+

Data enviada al receptor:  1010101111
Data modificada enviada al receptor:  0010101111

Resultado: Hubo un error en el bit 10
Se hizo la correcion. Trama final: 1010101111
```

Usando la trama 1:

```
Código de Hamming

Data enviada por el emisor: 1
Respuesta del emisor: 111

Archivo creado con éxito
```

```
+---+
| n |
+---+
| 0 |
+---+

Data enviada al receptor:  111

Resultado: Todo correcto, el mensaje no cuenta con errores
```

```
 +---+----+----+----+
 | n | P0 | P1 | P2 |
 +---+----+----+----+
 | 0 | 0  | 0  | 0  |
 | 1 | 1  | 0  | 0  |
 | 2 | 0  | 1  | 0  |
 | 3 | 1  | 1  | 0  |
 | 4 | 0  | 0  | 1  |
 | 5 | 1  | 0  | 1  |
 | 6 | 0  | 1  | 1  |
 | 7 | 1  | 1  | 1  |
 +---+----+----+----+

 Data enviada al receptor:  1010010
 Data modificada enviada al receptor:  1000010

 Resultado: Hubo un error en el bit 5
 Se hizo la correcion. Trama final: 1010010
```

Otros ejemplos de corrección de errores

```
 +---+----+----+----+
 | n | P0 | P1 | P2 |
 +---+----+----+----+
 | 0 | 0  | 0  | 0  |
 | 1 | 1  | 0  | 0  |
 | 2 | 0  | 1  | 0  |
 | 3 | 1  | 1  | 0  |
 | 4 | 0  | 0  | 1  |
 | 5 | 1  | 0  | 1  |
 | 6 | 0  | 1  | 1  |
 | 7 | 1  | 1  | 1  |
 +---+----+----+----+

 Data enviada al receptor:  1001100
 Data modificada enviada al receptor:  0001100

 Resultado: Hubo un error en el bit 7
 Se hizo la correcion. Trama final: 1001100
```

```
 +----+----+----+----+----+
 | n  | P0 | P1 | P2 | P3 |
 +----+----+----+----+----+
 | 0  | 0  | 0  | 0  | 0  |
 | 1  | 1  | 0  | 0  | 0  |
 | 2  | 0  | 1  | 0  | 0  |
 | 3  | 1  | 1  | 0  | 0  |
 | 4  | 0  | 0  | 1  | 0  |
 | 5  | 1  | 0  | 1  | 0  |
 | 6  | 0  | 1  | 1  | 0  |
 | 7  | 1  | 1  | 1  | 0  |
 | 8  | 0  | 0  | 0  | 1  |
 | 9  | 1  | 0  | 0  | 1  |
 | 10 | 0  | 1  | 0  | 1  |
 | 11 | 1  | 1  | 0  | 1  |
 +----+----+----+----+----+

 Data enviada al receptor:  1011010000
 Data modificada enviada al receptor:  0011010000

 Resultado: Hubo un error en el bit 10
 Se hizo la correcion. Trama final: 1011010000
```

Detección de errores:

Usando la trama 1001 a continuación se pueden observar todas las operaciones realizadas a la trama

```
con padding
10010000000000000000000000000000

polinomio
10110101000101010000111010011010

10010000000000000000000000000000
10110101000101010000111010011010
00100101000101010000111010011010

01001010000101010000111010011010

10010100001010100001110100110100
10110101000101010000111010011010
00100001010000010011010011110010

01000010100000100110100111100100
```

```
Resultado
10011000010100000100110100111100100
```

Archivo txt:

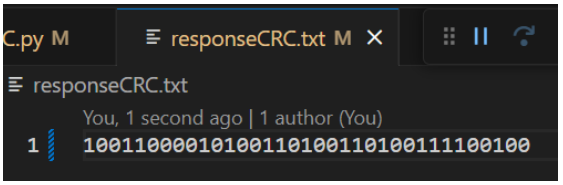
Operaciones realizadas por el receptor

```
10011000010100000100110100111100100
10110101000101010000111010011010
00101101010000101010000111010011010

01011010100001010100001110100110 1

10110101000101010000111010011010
10110101000101010000111010011010
00000000000000000000000000000000

00000000000000000000000000000000
```

Cuando la cadena ha sido pasada con éxito:

```
Resultado receptor
00000000000000000000000000000000
```

Trama modificada en el archivo



Luego de haber cambiado un bit en la secuencia en el archivo .txt

```
10011000010101100100110100111100100
10110101000101010000111010011010
00101101010000110100001110100110

01011010100001101000011101001101

10110101000011010000111010011010
10110101000101010000111010011010
00000000000110000000000000000000

00000000001100000000000000000000
```

```
Resultado receptor
00000010000000000000000000000000
```

Usando la trama 11101001010101010011010101101001, se consiguieron los siguientes resultados.

```
con padding
11101001010101010011010101101001000000
00000000000000000000000000

polinomio
10110101000101010000111010011010

11101001010101010011010101101001000000
00000000000000000000000000
10110101000101010000111010011010
01011100010000000011101111110011
```

```
Resultado
11101001010101010011010101101001100001
11010100000101001100101 10
Archivo creado con éxito
```


```
Resultado receptor
00000000000000000000000000000000
```

Usando la trama 11010010100101010010101010110100110

```
con padding
11010010100101010010101010110100110000000000000000000000000000000

polinomio
10110101000101010000111010011010

11010010100101010010101010110100110000000000000000000000000000000
10110101000101010000111010011010
01100011110000000010010000110111

Resultado
11010010100101010010101010110100110000111000111000100111010
Archivo creado con éxito
```


```
10110101000101010000111010011010
10110101000101010000111010011010
00000000000000000000000000000000

Resultado receptor
00000000000000000000000000000000
PS C:\Users\Michy S\Desktop\Michy\
```

Con la trama 110101

```
con padding
11010100000000000000000000000000000000

polinomio
10110101000101010000111010011010

11010100000000000000000000000000000000
10110101000101010000111010011010
01100001000101010000111010011010

Resultado
11010100011110000110101110100010110000
Archivo creado con éxito
```


```
10110101000101010000111010011010
10110101000101010000111010011010
00000000000000000000000000000000

00000000000000000000000000000000

00000000000000000000000000000000

Resultado receptor
00000000000000000000000000000000
```

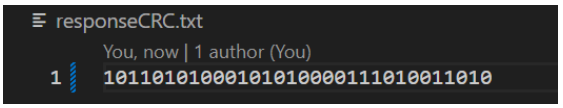
Usando la trama 1010010100100

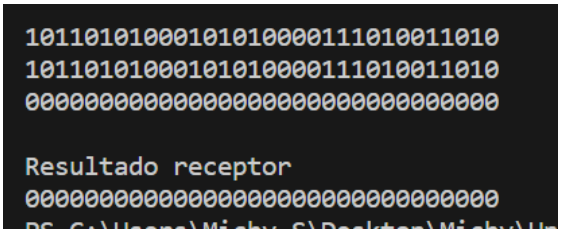
```
con padding
10100101001000000000000000000000000000000000

polinomio
10110101000101010000111010011010

10100101001000000000000000000000000000000000
10110101000101010000111010011010
00010000001101010000111010011010

Resultado
10100101001001001101101010100110110010100110
Archivo creado con éxito
```


```
10110101000101010000111010011010
10110101000101010000111010011010
00000000000000000000000000000000

Resultado receptor
00000000000000000000000000000000
PS C:\Users\Michy S\Desktop\Michy\Ur
```

Usando la trama 1

```
con padding
10000000000000000000000000000000

polinomio
10110101000101010000111010011010

10000000000000000000000000000000
10110101000101010000111010011010
00110101000101010000111010011010

Resultado
10110101000101010000111010011010
Archivo creado con éxito
```





Tramas de prueba:

- 1001 (la que se utilizó en clase)
- 11101001010101010011010101101001
- 1101001010010101001010101010110100110
- 1010010100100
- 110101
- 1
- Entre otras con la finalidad de mostrar descartes de trama, detección y corrección de errores

Discusión

En el presente laboratorio se buscó comprender mejor los conceptos y técnicas para la detección y corrección de errores en mensajes ya que en un proceso de comunicación puede existir factores que distorsionen, arruinen o impidan que los mensajes lleguen a su destino de manera adecuada.

De los resultados obtenidos podemos observar que del 100% de pruebas realizadas un 90% fueron exitosas, los problemas se dieron en la implementación del algoritmo de hamming, ya que este en cadenas demasiados largas no terminaba de determinar si la cadena poseía errores o no. Esto se pudo deber a que la implementación del código, del lado del receptor no es del todo exacta al calcular los conjuntos de bits que permiten determinar si hay o no errores y de haberlos el conjunto de bits encontrados no se convierte correctamente en el decimal que hace referencia a la posición que debería representar.

El algoritmo Hamming es utilizado para la corrección de errores al momento de transmitir, tal como se puede observar en la implementación realizada. A diferencia del CRC-32 se introduce una mejora significativa en la capacidad del sistema para que este pueda detectar y corregir los errores que se pueden producir en la transmisión de datos, ya que no solo detecta sino que corrige cualquier dato al cual le haya ocurrido un flip, de esa manera, recuperando datos dañados y asegurando una mayor fiabilidad en el momento de la comunicación y conexión de redes.

Cuando el mensaje es emitido por el programa de java, este algoritmo agrega bits adicionales de paridad en posiciones estratégicas dentro de los datos. Estos bits extra permiten identificar con mayor facilidad los switches porque si esto ocurre luego estas paridades no cuadran. Esta

parte del trabajo es tarea del receptor en Python. Si detecta que contiene errores, Hamming realiza una operación de corrección y recupera los datos originales sin la necesidad de volver a transmitir la información. Permitiendo así garantizar una mayor integridad y confiabilidad de los datos recibidos.

En el caso de la implementación realizada para el presente laboratorio, es muy poco probable que los datos hayan sido almacenados y transmitidos de manera incorrecta, por lo que para probar su funcionamiento simplemente se hicieron switches manualmente en la cadena transmitida, y el algoritmo correctamente logró determinar y corregir estos switches.

Por otro lado, el algoritmo CRC, tal como ha sido utilizado en esta implementación, es una técnica que se utiliza para detectar errores durante la transmisión de datos en redes y en sistemas de almacenamiento el cual facilita su uso en sistemas heterogéneos. Tal como fue especificado anteriormente, la implementación consta de un emisor y de un receptor los cuales fueron escritos en java y python respectivamente. Gracias a que estos fueron escritos en 2 lenguajes de programación distintos se puede demostrar que es un algoritmo portable e interoperable.

Es importante recordar que el algoritmo CRC-32 es un un procedimiento matemático, el cual genera un valor de verificación único, conocido como “checksum” el cual en este caso fue almacenado en un archivo de texto luego de la trama original. Luego el receptor leyó esta trama extrayendo el checksum para recalcular el CRC-32 y verificar si el valor obtenido coincidía con el checksum proporcionado por el emisor. Si este daba un resultado sin residuos, este no había sido modificado durante la transmisión. Eso se puede observar con todos los residuos que han quedado solo como arreglos de 0. En el caso contrario, y existe cualquier 1 en la cadena, ha ocurrido un error. A diferencia del otro algoritmo implementado para este laboratorio, este no puede determinar correctamente cuáles son los bytes que realizaron flip para poder hacer sus respectivas correcciones. Por lo que de las pocas acciones que se pueden hacer luego de detectar un problema con la transmisión, es volverlo a enviar.

Comentarios

- Iniciar el laboratorio fue algo difícil, especialmente por la falta de comprensión de las instrucciones.
    - Fue un poco confuso al principio y se desarrolló más código del solicitado buscando completar el lab de la forma en que habíamos entendido las instrucciones, esto nos llevó a realizar diferentes versiones del código, diversas pruebas y la cantidad de pruebas solicitadas representaba mucho tiempo para su documentación.
- La implementación de Hamming en java fue más costosa que la implementación en Java debido a que no podía devolver múltiples objetos o parámetros de un mismo método y el manejo de “diccionarios” dificultó el desarrollo y el manejo de las

posiciones de los bits en diferentes estructuras de datos se volvió un poco más compleja de lo esperado.

- El receptor de hamming no se implementó con un 100% de aceptación de cadenas ya que este fallaba en cadenas muy largas en el sentido de que no lograba determinar si estaba bien o mal. Entre las debilidades que se investigaron se encontró que para cadenas más pequeñas, la probabilidad de determinar si había errores también disminuye, está limitado a un bit incorrecto por lo que si en pruebas modificamos 2 o más no se encontraban errores y al ir acumulando bits de paridad para determinar información necesaria que indique si hay errores o no se puede confundir el algoritmo y dar un falso positivo.

Conclusiones

- Se puede concluir que se logró realizar una implementación exitosa de los algoritmos Hamming y CRC-32 en los lenguajes de programación Java y Python.
- El algoritmo de Hamming representa una solución relativamente sencilla de implementar para la identificación y corrección de errores aunque ante ciertas cadenas puede no resultar 100% efectivo dando falsos positivos en el procesamiento de mensajes.
- El algoritmo de CRC-32 es un algoritmo sencillo de implementar el cual determina efectivamente cuando una trama ha sido alterada. Este ayuda a verificar la integridad del entorno de transmisión, y refuerza la importancia de la interoperabilidad entre distintos lenguajes de programación.