UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería
CC3085 – Redes
Sección 10
Ing. Miguel Novella Linare

Laboratorio 2

Samuel Argueta - 211024
Alejandro Martinez - 21430

GUATEMALA, 25 de julio de 2024

REPOSITORIO

https://github.com/Gustixa/redes.git

CRC-32

Emisor: Java
Receptor: python

Escenario sin error

Mensaje original: 1010101111001101
Mensaje enviado: 10101011110011010011011011001011111001000111010 1
Mensaje recibido: 10101011110011010011011011001011111001000111010 1

```
arg@arg-G7-7588:~/Documents/uvg/semestre8/redes/lab2/receptor$  cd /home/arg/Documents/uv
va -XX:+ShowCodeDetailsInExceptionMessages -cp /home/arg/Documents/uvg/semestre8/redes/la
CRC generated: 00110110110010111110010001110101
Codeword transmitted: 10101011110011010011011011001011111001000111010 1
```

```
arg@arg-G7-7588:~/Documents/uvg/semestre8/redes/lab2$ python3 crc.py
Ingrese el código recibido: 10101011110011010011011011001011111001000111010 1
No hay error en la transmisión
```

Se puede apreciar que en este caso, el resultado fue el esperado, pues no hubo error en el codigo enviado, ni se percibió alguno, cuando se ingresó en el receptor.

Escenario con 1 error, cambiando el último bit por su inverso, en este caso, 0 → 1

Mensaje original: 110010101111
Mensaje enviado: 1100101011110000011000001100100010010100 1000
Mensaje enviado con error: 1100101011110000011000001100100010010100 1001

```
arg@arg-G7-7588:~/Documents/uvg/semestre8/redes/lab2/receptor$  cd /h
va -XX:+ShowCodeDetailsInExceptionMessages -cp /home/arg/Documents/uv
CRC generated: 00000110000011001000100101001000
Codeword transmitted: 11001010111100000110000011001000100101001000
```

```
arg@arg-G7-7588:~/Documents/uvg/semestre8/redes/lab2$ python3 crc.py
Ingrese el código recibido: 11001010111100000110000011001000100101001001
Error en la transmisión
```

Escenario con 2 o más errores, cambiando el primer, último y penúltimo bit:

Mensaje original: 100110111
Mensaje enviado: 1001101110001100001101100111111 10001001001
Mensaje enviado con errores: 0001101110001100001101100111111 10001001010

```
arg@arg-G7-7588:~/Documents/uvg/semestre8/redes/lab2/receptor$  cd
va -XX:+ShowCodeDetailsInExceptionMessages -cp /home/arg/Documents
CRC generated: 00011000011011001111110001001001
Codeword transmitted: 10011011100011000011011001111110001001001
```

```
arg@arg-G7-7588:~/Documents/uvg/semestre8/redes/lab2$ python3 crc.py
Ingrese el código recibido: 0001101110001100001101100111111000100101O
Error en la transmisión
```

En ambos casos, fue capaz de detectar el error que se generó de manera maliciosa/a propósito demostrando que es verdad, sirve para la detección de los mismos.

HAMMING GENERICO

Emisor:
Receptor:
Mensaje original: 1010101111001101
Mensaje enviado: 1010010010111100
Mensaje recibido: 10101011110

```
PS D:\UVG\Redes\redes\Lab 02\output> & .\'Encoder.exe'
Enter the total number of bits (n): 16
Enter the number of data bits (k): 16
Enter a binary message of 16 bits: 1010101111001101
Encoded message: 1010010010111100
```

```
 "d:/UVG/Redes/redes/Lab 02/Decoder.py"
Enter the encoded message: 1010010010111100
Enter the total number of bits (n): 16
Enter the number of data bits (k): 16
No errors detected.
Original Corrected message: 10101011110
```

Escenario con 1 error, cambiando el último bit por su inverso, en este caso, 0 → 1
Mensaje original: 110010101111
Mensaje enviado: 101010001010111
Mensaje enviado con error: 000010101111
Mensaje recibido: 11001010111
Mensaje recibido con error: 000010101111

```
PS D:\UVG\Redes\redes\Lab 02\output> & .\'Encoder Modified.exe'
Enter the total number of bits (n): 15
Enter the number of data bits (k): 12
Enter a binary message of 12 bits: 110010101111
Encoded message: 101010001010111
Modified message: 000010101111
```

```
 "d:/UVG/Redes/redes/Lab 02/Decoder.py"
Enter the encoded message: 101010001010111
Enter the total number of bits (n): 15
Enter the number of data bits (k): 12
Detected error at position: 1
Corrected bits: 001010001010111
Original Corrected message: 11001010111
```

```
 "d:/UVG/Redes/redes/Lab 02/Decoder.py"
Enter the encoded message: 000010101111
Enter the total number of bits (n): 12
Enter the number of data bits (k): 12
No errors detected.
Original Corrected message: 01011111
```

CUESTIONARIO FINAL

¿qué ventajas y desventajas posee cada algoritmo con respecto a los otros dos? Tome en cuenta complejidad, velocidad, redundancia (overhead), etc.

	Hamming	CRC-32
Complejidad	Son más simples de implementar, pues requieren menos operaciones matemáticas que CRC-32.	Es más complejo y requiere más operaciones de desplazamiento y XOR, pero es mejor en detección de errores.

Velocidad	Pueden ser calculados y corregidos rápidamente, pero su capacidad de corrección es limitada a 1 bit.	Se beneficia mucho más en las optimizaciones del hardware.
Overhead	Hay menos overhead, en casos como (11,7) y (7,4) es más notorio.	El overhead es fijo, depende de cual se este implementando (CRC-8, 16 o 32) Si es CRC-32, el n es de 32 lo que se puede considerar alto para mensajes cortos.
Detección y Corrección de errores	Pueden detectar y corregir errores de 1 solo bit, y detectar errores de 2 bits.	Es bueno en la detección de errores, incluyendo los de rafaga, pero no puede corregir.

Al final de cuentas, ambos algoritmos son útiles, pero es recomendable tomar consideraciones en cuanto a cual implementar y porque, pues como se plantea en el cuadro comparativo, cada uno tiene sus pros y contras y dependiendo del contexto, uno sirve más que otro, o bien, uno es demasiado para tareas simples. Para aplicaciones donde la corrección de errores es crucial y los recursos son limitados, los códigos Hamming pueden ser más adecuados. Para aplicaciones donde la detección de errores es más crítica y se requiere alta robustez contra errores de ráfaga, CRC-32 es la mejor opción.