

Laboratorio 2 Reporte

Pruebas

Hamming

Sin errores:

```
PS C:\Users\garci\OneDrive\Documentos\Tercer semestre U\IALab4\Lab2Redes> & C:\Users\garci\AppData\Local\Microsoft\Windows\Apps\python.exe "c:/Users/garci/OneDrive/Documentos/Tercer semestre U/IALab4/Lab2Redes/emitter.py"
Ingrese la trama original en binario (ej. 1011001): 1011
Trama codificada (Hamming): 0110011
PS C:\Users\garci\OneDrive\Documentos\Tercer semestre U\IALab4\Lab2Redes>

PS C:\Users\garci\OneDrive\Documentos\Tercer semestre U\IALab4\Lab2Redes> node receiver.js
Ingrese la trama codificada (Hamming): 0110011
No se detectaron errores.
Trama recibida (corregida si aplica): 0110011
Datos extraídos: 1011
PS C:\Users\garci\OneDrive\Documentos\Tercer semestre U\IALab4\Lab2Redes>

Ingrese la trama original en binario (ej. 1011001): 0110011
Trama codificada (Hamming): 01001100011
PS C:\Users\garci\OneDrive\Documentos\Tercer semestre U\IALab4\Lab2Redes>

Ingrese la trama codificada (Hamming): 01001100011
No se detectaron errores.
Trama recibida (corregida si aplica): 01001100011
Datos extraídos: 0110011
PS C:\Users\garci\OneDrive\Documentos\Tercer semestre U\IALab4\Lab2Redes>

Ingrese la trama original en binario (ej. 1011001): 01101110
Trama codificada (Hamming): 110011011110
PS C:\Users\garci\OneDrive\Documentos\Tercer semestre U\IALab4\Lab2Redes>

Ingrese la trama codificada (Hamming): 110011011110
No se detectaron errores.
Trama recibida (corregida si aplica): 110011011110
Datos extraídos: 01101110
```

Un error:

Original 1011 → 0110011

Mutar bit 3 (cód. recibido = 0100011)

```
/Programs/Python/Python313/python.exe "c:/Users/garci/OneDrive/Documentos/Tercer semestre U/IALab4/Lab2Redes/emitter.py"
Ingrese la trama original en binario (ej. 1011001): 1011
Trama codificada (Hamming): 0110011

Ingrese la trama codificada (Hamming): 0100011
Error detectado en posición 3. Corrigiendo...
Trama recibida (corregida si aplica): 0110011
Datos extraídos: 1011
```

Original 1101001 → 01101011001

Mutar bit 5 (cód. recibido = 01100011001)

```
/Programs/Python/Python313/python.exe "c:/Users/garci/OneDrive/Documentos/Tercer semestre U/IALab4/Lab2Redes/emitter.py"
Ingrese la trama original en binario (ej. 1011001): 1101001
Trama codificada (Hamming): 01101011001

Ingrese la trama codificada (Hamming): 01100011001
Error detectado en posición 5. Corrigiendo...
Trama recibida (corregida si aplica): 01101011001
Datos extraídos: 1101001
```

Original 01101110 → 110011011110

Mutar bit 2 (cód. recibido = 100011011110)

```
/Programs/Python/Python313/python.exe "c:/Users/garci/OneDrive/Documentos/Tercer semestre U/IALab4/Lab2Redes/emitter.py"
Ingrese la trama original en binario (ej. 1011001): 01101110
Trama codificada (Hamming): 110011011110

Ingrese la trama codificada (Hamming): 100011011110
Error detectado en posición 2. Corrigiendo...
Trama recibida (corregida si aplica): 110011011110
Datos extraídos: 01101110
```

Dos errores o más:

Hamming sólo asegura corrección de un bit. Con dos o más, el receptor trata de corregir “un” error, pero el resultado queda incorrecto:

Original 1011 → 0110011

Mutar bits 2 y 5 → recibido 0010111

```
/Programs/Python/Python313/python.exe "c:/Users/garci/OneDrive/Documentos/Tercer semestre U/IALab4/Lab2Redes/emitter.py"
Ingrese la trama original en binario (ej. 1011001): 1011
Trama codificada (Hamming): 0110011

Ingrese la trama codificada (Hamming): 0010111
Error detectado en posición 7. Corrigiendo...
Trama recibida (corregida si aplica): 0010110
Datos extraídos: 1110
```

Sofía García – 22210
Julio García Salas - 22076

No logra corregirlo por la naturaleza del algoritmo de Hamming

Original 1101001 → 01101011001

Mutar bits 3 y 4 → recibido 01011011001

```
/Programs/Python/Python313/python.exe "c:/Users/garci/OneDrive/Documentos/Tercer semestre U/IALab4/Lab2Redes/emisor.py"
Ingrese la trama original en binario (ej. 1011001): 1101001
Trama codificada (Hamming): 01101011001

Ingrese la trama codificada (Hamming): 01101011001
Error detectado en posición 7. Corrigiendo...
Trama recibida (corregida si aplica): 01101001001
Datos extraídos: 0100001
```

Original 011011110 → 110011011110

Mutar bits 5 y 6 → recibido 110000011110

```
/Programs/Python/Python313/python.exe "c:/Users/garci/OneDrive/Documentos/Tercer semestre U/IALab4/Lab2Redes/emisor.py"
Ingrese la trama original en binario (ej. 1011001): 01101110
Trama codificada (Hamming): 110011011110

Ingrese la trama codificada (Hamming): 110000011110
Error detectado en posición 3. Corrigiendo...
Trama recibida (corregida si aplica): 111000011110
Datos extraídos: 10001110
```

CRC-32

Sin errores:

```
Ingrese la trama de bits (ej. 110101): 110101
Trama codificada con CRC-32:
11010111000001111101110000011011111011

111101110000011011111011
CRC VÁLIDO ✓
Datos originales: 110101

Ingrese la trama de bits (ej. 110101): 10110011
Trama codificada con CRC-32:
1011001101100000111101101010110100111

0000111101101010110100111
CRC VÁLIDO ✓
Datos originales: 10110011

/Programs/Python/Python313/python.exe "c:/Users/garci/OneDrive/Documentos/Tercer semestre U/IALab4/Lab2Redes/emisor/crc_emisor.py"
Ingrese la trama de bits (ej. 110101): 0110111001
Trama codificada con CRC-32:
01101110010100110111011010111001010101

01101110111010111001010101011011100101001101110110101110001010101
CRC VÁLIDO ✓
Datos originales: 01101110010100110111010101110010101010101010101
PS C:\Users\garci\OneDrive\Documentos\Tercer semestre U\IALab4\Lab2Redes>
```

Un error:

Original Codificada: 1101011100001111101110000011011111011

Flipeado: 1001011100001111101110000011011111011

```
PS C:\Users\garci\OneDrive\Documentos\Tercer semestre U\IALab4\Lab2Redes> & C:/Users/garci/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/garci/OneDrive/Documentos/Tercer semestre U/IALab4/Lab2Redes/emisor/crc_emisor.py"
Ingrese la trama de bits (ej. 110101): 110101
Trama codificada con CRC-32:
110101110000011111011010101110111011

PS C:\Users\garci\OneDrive\Documentos\Tercer semestre U\IALab4\Lab2Redes> node CrcReceiver 1001011100001111101110000011011111011
11110111000001101111011
ERROR DE CRC ✗ - trama corrupta o mal ingresada.
PS C:\Users\garci\OneDrive\Documentos\Tercer semestre U\IALab4\Lab2Redes>
```

Original Codificada: 1011001110110000011111011010101110100111

Flipeado: 1111001110110000011111011010101110100111

```
Ingrese la trama de bits (ej. 110101): 10110011
Trama codificada con CRC-32:
10110011011000001111011010101110100111

PS C:\Users\garci\OneDrive\Documentos\Tercer semestre U\IALab4\Lab2Redes> node CrcReceiver 11110011101101100111001110110000011111011010101110100111
00001111011010101110100111
ERROR DE CRC ✗ - trama corrupta o mal ingresada.
```

Original Codificada: 01101110010100110111101110101110010101101

Flipeado: 00101110010100110111101110101110010101101

```
PS C:\Users\garci\OneDrive\Documentos\Tercer semestre U\IALab4\Lab2Redes> & C:/Users/garci/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/garci/OneDrive/Documentos/Tercer semestre U/IALab4/Lab2Redes/emisor/crc_emisor.py"
Ingrese la trama de bits (ej. 110101): 0110111001
Trama codificada con CRC-32:
0110111001010011011110111010111001010101

PS C:\Users\garci\OneDrive\Documentos\Tercer semestre U\IALab4\Lab2Redes> node CrcReceiver 00101110010101100111001110110000011111011010101110100111
011011110110101110010101010110111001010101010101
ERROR DE CRC ✗ - trama corrupta o mal ingresada.
PS C:\Users\garci\OneDrive\Documentos\Tercer semestre U\IALab4\Lab2Redes>
```

Dos errores:

Original Codificada: 1101011100001111101110000011011111011

Flipeado: 0111011100001111101110000011011111011

Sofía García – 22210
Julio García Salas - 22076

```
PS C:\Users\garci\OneDrive\Documentos\Tercer semestre U\IALab4\Lab2Redes> & C:\Users\garci\AppData\Local\
/Programs/Python/Python313/python.exe "c:/Users/garci/OneDrive/Documentos/Tercer semestre U/IALab4/Lab2R
edes/crc_emitter.py"
Ingrese la trama de bits (ej. 110101): 110101
Trama codificada con CRC-32:
1101011100001111101100000111111011010101110100111

PS C:\Users\garci\OneDrive\Documentos\Tercer semestre U\IALab4\Lab2Redes> node CrcReceiver 0111011100001
11110111000001101111011
ERROR DE CRC X - trama corrupta o mal ingresada.
❖ PS C:\Users\garci\OneDrive\Documentos\Tercer semestre U\IALab4\Lab2Redes> []
```

Original Codificada: 1011001110110000011111011010101110100111

Flipeado: 0001001110110000011111011010101110100111

```
PS C:\Users\garci\OneDrive\Documentos\Tercer semestre U\IALab4\Lab2Redes> & C:\Users\garci\AppData\Local\
/Programs/Python/Python313/python.exe "c:/Users/garci/OneDrive/Documentos/Tercer semestre U/IALab4/Lab2R
edes/crc_emitter.py"
Ingrese la trama de bits (ej. 110101): 10110011
Trama codificada con CRC-32:
101100111001001101110110101110100111

PS C:\Users\garci\OneDrive\Documentos\Tercer semestre U\IALab4\Lab2Redes> node CrcReceiver 00010011101010
11110111000001101110111
ERROR DE CRC X - trama corrupta o mal ingresada.
❖ PS C:\Users\garci\OneDrive\Documentos\Tercer semestre U\IALab4\Lab2Redes> []
```

Original Codificada: 011011100101001101111011101011110010101101

Flipeado: 110011100101001101111011101011110010101101

```
PS C:\Users\garci\OneDrive\Documentos\Tercer semestre U\IALab4\Lab2Redes> & C:\Users\garci\AppData\Local\
/Programs/Python/Python313/python.exe "c:/Users/garci/OneDrive/Documentos/Tercer semestre U/IALab4/Lab2R
edes/crc_emitter.py"
Ingrese la trama de bits (ej. 110101): 0110111001
Trama codificada con CRC-32:
0110111001001001101110101110010101101

PS C:\Users\garci\OneDrive\Documentos\Tercer semestre U\IALab4\Lab2Redes>

PS C:\Users\garci\OneDrive\Documentos\Tercer semestre U\IALab4\Lab2Redes> node CrcReceiver 110011100101010
11110111000001101110111
ERROR DE CRC X - trama corrupta o mal ingresada.
❖ PS C:\Users\garci\OneDrive\Documentos\Tercer semestre U\IALab4\Lab2Redes> []
```

¿Es posible manipular los bits de tal forma que el algoritmo seleccionado no sea capaz de detectar el error? ¿Por qué sí o por qué no? En caso afirmativo, demuestrelo con su implementación

Es posible manipular los bits de tal forma que el algoritmo de Hamming no sea capaz de detectar un error múltiple, ya que el esquema de Hamming estándar está diseñado para corregir un solo bit erróneo y puede confundir dos o más errores con uno solo. En nuestra implementación, cuando alteramos dos bits en posiciones específicas de la trama codificada, el receptor calcula un síndrome distinto de cero, interpreta que hay un único error en otra posición y “corrige” ese bit, sin advertir la presencia de un segundo error. De este modo, el mensaje final queda corrupto sin que el protocolo lo detecte como fallo irremediable, demostrando que Hamming puro carece de un mecanismo de alarma para errores de ráfaga o múltiples.

Comparación entre Hamming y los otros algoritmos

En base a las pruebas realizadas (que incluyeron únicamente el código de Hamming) y a la investigación bibliográfica de otros métodos, podemos identificar las siguientes ventajas y desventajas. El bit de paridad par ofrece la redundancia más baja (solo un bit extra) y una implementación ultrarrápida gracias a una sola operación XOR, pero solo detecta errores de paridad impar y no corrige ninguno, siendo incapaz de advertir dos errores concurrentes. Hamming añade un overhead logarítmico en bits de paridad para corregir automáticamente un único error y detectar cuándo hay al menos dos, aunque no puede diferenciar dos fallos de uno solo, y su complejidad aumenta con el número de rutas de paridad a verificar. CRC-32, basándonos en la literatura, inserta 32 bits de redundancia y emplea aritmética polinómica para detectar con alta fiabilidad ráfagas largas, dobles o triples errores, pero no corrige fallos y supone un mayor coste de procesamiento para el cálculo del residuo. En definitiva, la elección del método dependerá del tipo de canal, la tolerancia al overhead y la necesidad de corrección frente a detección pura.

¿Es posible manipular los bits de tal forma que el algoritmo seleccionado no sea capaz de detectar el error? ¿Por qué sí o por qué no? En caso afirmativo, demuéstrela con su implementación.

En el caso del algoritmo CRC-32, no es fácil manipular los bits de una trama de tal forma que el error pase desapercibido, debido a su alto nivel de confiabilidad en la detección de errores. CRC-32 está diseñado para detectar errores simples, dobles, errores en ráfaga y cambios en múltiples bits, siempre que la alteración no sea intencionadamente construida para burlar el algoritmo. A diferencia de Hamming, CRC-32 no intenta corregir los errores, sino únicamente detectarlos. Durante nuestras pruebas, alteramos un bit, y luego dos o más bits, en tramas codificadas, y en todos los casos el receptor detectó correctamente que el CRC era inválido, descartando la trama. Sin embargo, teóricamente es posible construir una segunda trama con el mismo residuo CRC (una *colisión*), pero hacerlo requiere conocimiento avanzado del polinomio y no ocurre por casualidad. Por tanto, para efectos prácticos, CRC-32 es extremadamente confiable como mecanismo de detección.

Comparación entre CRC-32 y los otros algoritmos

En base a las pruebas realizadas con CRC-32 y a la investigación sobre otros algoritmos, se identifican varias diferencias clave. El bit de paridad par es el más simple de implementar, con solo un bit adicional de redundancia y operaciones XOR básicas, pero solo detecta errores cuando la cantidad de bits alterados es impar, lo que limita su confiabilidad.

El código de Hamming, por su parte, añade algunos bits de paridad en posiciones específicas para detectar y corregir errores de un solo bit, y detectar errores de dos bits, aunque no los puede corregir. Su ventaja está en que permite recuperación parcial del mensaje, aunque no distingue entre uno y múltiples errores.

CRC-32, en contraste, introduce un mayor overhead fijo de 32 bits, pero lo compensa con una capacidad de detección mucho más robusta. Está especialmente diseñado para detectar errores de ráfaga, múltiples alteraciones aleatorias y secuencias modificadas, gracias a su modelo de división polinómica.

Su desventaja principal es que no corrige errores y que requiere más procesamiento que los otros métodos, pero es ideal para entornos donde la confiabilidad en la detección es prioritaria. En resumen, CRC-32 es el algoritmo más confiable para detectar errores, aunque no el más ligero ni el más simple.