

UNIVERSIDAD DEL VALLE DE GUATEMALA

## REDES



## Laboratorio 2

Guatemala, Agosto de 2025

Sebastián Huertas 22295

Josué Marroquín 22397

## Primera parte - Esquema de detección y corrección de errores

### Escenarios prueba

Mensajes a probar 10000, 10101010011, 010101010101010100101

### Sin errores:

Enviar un mensaje al emisor, copiar el mensaje generado por este y proporcionarlo tal cual, al receptor, el cual debe mostrar los mensajes originales (ya que ningún bit sufrió un cambio). Realizar esto para tres mensajes distintos con distinta longitud.

Fletcher-16

<pre>PS C:\Users\josue\uvg\Semestre 8\Redes\data-link-layer-error-handling\emisor_ts&gt; npm start  &gt; emisor_ts@1.0.0 start &gt; tsx main.ts  === EMISOR ===  Ingrese la cadena de binario: 10000  Cadena binaria ingresada: 10000  === Seleccione el algoritmo === 1. Hamming 2. Fletcher-16 Ingrese su opcion (1 o 2): 3 Error: Ingrese 1 o 2. Intente nuevamente. Ingrese su opcion (1 o 2): 3 Error: Ingrese 1 o 2. Intente nuevamente. Ingrese su opcion (1 o 2): 2  === Algoritmo Fletcher-16 seleccionado === Mensaje escrito exitosamente en: ../tests/mensaje.txt PS C:\Users\josue\uvg\Semestre 8\Redes\data-link-layer-error-handling\emisor_ts&gt;</pre>	<pre>PS C:\Users\josue\uvg\Semestre 8\Redes\data-link-layer-error-handling\receptor_py&gt; python main.py  === RECEPTOR ===  === Seleccione el algoritmo de recepción === 1. Hamming 2. Fletcher-16 Ingrese su opción (1 o 2): 2 Mensaje recibido: 100001000000010000000  === Algoritmo Fletcher-16 seleccionado === No se detectaron errores. Mensaje original: 10000 PS C:\Users\josue\uvg\Semestre 8\Redes\data-link-layer-error-handling\receptor_py&gt;</pre>
<pre>&gt; emisor_ts@1.0.0 start &gt; tsx main.ts  === EMISOR ===  Ingrese la cadena de binario: 0101010101010100101  Cadena binaria ingresada: 0101010101010100101  === Seleccione el algoritmo === 1. Hamming 2. Fletcher-16 Ingrese su opcion (1 o 2): 2  === Algoritmo Fletcher-16 seleccionado === Mensaje escrito exitosamente en: ../tests/mensaje.txt PS C:\Users\josue\uvg\Semestre 8\Redes\data-link-layer-error-handling\emisor_ts&gt;</pre>	<pre>=== RECEPTOR ===  === Seleccione el algoritmo de recepción === 1. Hamming 2. Fletcher-16 Ingrese su opción (1 o 2): 2 Mensaje recibido: 01010101010101001011101001011010010  === Algoritmo Fletcher-16 seleccionado === No se detectaron errores. Mensaje original: 01010101010100101 PS C:\Users\josue\uvg\Semestre 8\Redes\data-link-layer-error-handling\receptor_py&gt;</pre>
<pre>&gt; emisor_ts@1.0.0 start &gt; tsx main.ts  === EMISOR ===  Ingrese la cadena de binario: 10101010011  Cadena binaria ingresada: 10101010011  === Seleccione el algoritmo === 1. Hamming 2. Fletcher-16 Ingrese su opcion (1 o 2): 2  === Algoritmo Fletcher-16 seleccionado === Mensaje escrito exitosamente en: ../tests/mensaje.txt PS C:\Users\josue\uvg\Semestre 8\Redes\data-link-layer-error-handling\emisor_ts&gt;</pre>	<pre>=== RECEPTOR ===  === Seleccione el algoritmo de recepción === 1. Hamming 2. Fletcher-16 Ingrese su opción (1 o 2): 2 Mensaje recibido: 101010100111011010100001011  === Algoritmo Fletcher-16 seleccionado === No se detectaron errores. Mensaje original: 10101010011 PS C:\Users\josue\uvg\Semestre 8\Redes\data-link-layer-error-handling\receptor_py&gt;</pre>

## Hamming

```
Command Prompt
C:\Users\thiag\Universidad\data-link-layer-error-handling\emisor_ts
>npm start
> emisor_ts@1.0.0 start
> tsx main.ts

=== EMISOR ===

Ingrese la cadena de binario: 10000

Cadena binaria ingresada: 10000

=== Seleccione el algoritmo ===
1. Hamming
2. Fletcher-16
Ingrese su opcion (1 o 2): 1

=== Algoritmo Hamming seleccionado ===
Bits de paridad: 4
Posiciones con bits de paridad (en 0): 0,0,1,0,0,0,0,0
Nuevo mensaje: 111000000
Mensaje escrito exitosamente en: ../tests/mensaje.txt

C:\Users\thiag\Universidad\data-link-layer-error-handling\emisor_ts
>

Command Prompt
C:\Users\thiag\Universidad\data-link-layer-error-handling\receptor_
py>python main.py
=== RECEPTOR ===

=== Seleccione el algoritmo de recepción ===
1. Hamming
2. Fletcher-16
Ingrese su opción (1 o 2): 1
Mensaje recibido: 111000000

=== Algoritmo Hamming seleccionado ===
Procesando mensaje Hamming (receptor): 111000000
posiciones de los bits de paridad: [1, 2, 4, 8]
No se detectaron errores.
Mensaje original: 10000

C:\Users\thiag\Universidad\data-link-layer-error-handling\receptor_
py>
```

```
Command Prompt
C:\Users\thiag\Universidad\data-link-layer-error-handling\emisor_ts
>npm start
> emisor_ts@1.0.0 start
> tsx main.ts

=== EMISOR ===

Ingrese la cadena de binario: 10101010011

Cadena binaria ingresada: 10101010011

=== Seleccione el algoritmo ===
1. Hamming
2. Fletcher-16
Ingrese su opcion (1 o 2): 1

=== Algoritmo Hamming seleccionado ===
Bits de paridad: 4
Posiciones con bits de paridad (en 0): 0,0,1,0,0,1,0,0,1,0,1,0,1,
1
Nuevo mensaje: 011101001010011
Mensaje escrito exitosamente en: ../tests/mensaje.txt

C:\Users\thiag\Universidad\data-link-layer-error-handling\emisor_ts
>

Command Prompt
C:\Users\thiag\Universidad\data-link-layer-error-handling\receptor_
py>python main.py
=== RECEPTOR ===

=== Seleccione el algoritmo de recepción ===
1. Hamming
2. Fletcher-16
Ingrese su opción (1 o 2): 1
Mensaje recibido: 011101001010011

=== Algoritmo Hamming seleccionado ===
Procesando mensaje Hamming (receptor): 011101001010011
posiciones de los bits de paridad: [1, 2, 4, 8]
No se detectaron errores.
Mensaje original: 10101010011

C:\Users\thiag\Universidad\data-link-layer-error-handling\receptor_
py>
```

```
Command Prompt
C:\Users\thiag\Universidad\data-link-layer-error-handling\emisor_ts
>npm start
> emisor_ts@1.0.0 start
> tsx main.ts

=== EMISOR ===

Ingrese la cadena de binario: 0101010101010100101

Cadena binaria ingresada: 0101010101010100101

=== Seleccione el algoritmo ===
1. Hamming
2. Fletcher-16
Ingrese su opcion (1 o 2): 1

=== Algoritmo Hamming seleccionado ===
Bits de paridad: 5
Posiciones con bits de paridad (en 0): 0,0,0,1,0,1,0,0,1,0,1,0,1,
0,0,1,0,1,0,1,0,0,1,0,1
Nuevo mensaje: 11011011010101010100101
Mensaje escrito exitosamente en: ../tests/mensaje.txt

C:\Users\thiag\Universidad\data-link-layer-error-handling\emisor_ts
>

Command Prompt
C:\Users\thiag\Universidad\data-link-layer-error-handling\receptor_
py>python main.py
=== RECEPTOR ===

=== Seleccione el algoritmo de recepción ===
1. Hamming
2. Fletcher-16
Ingrese su opción (1 o 2): 1
Mensaje recibido: 11011011010101010100101

=== Algoritmo Hamming seleccionado ===
Procesando mensaje Hamming (receptor): 11011011010101010100101
posiciones de los bits de paridad: [1, 2, 4, 8, 16]
No se detectaron errores.
Mensaje original: 0101010101010100101

C:\Users\thiag\Universidad\data-link-layer-error-handling\receptor_
py>
```

Un error:

Enviar un mensaje al emisor, copiar el mensaje generado por este y cambiar un bit cualquiera antes de proporcionarlo al receptor. Si el algoritmo es de detección debe mostrar que se detectó un error y que se descarta el mensaje. Si el algoritmo es de corrección debe corregir el bit, indicar su posición y mostrar el mensaje original. Realizar esto para tres mensajes distintos con distinta longitud.

## Fletcher-16

Entrada emisor: 10000

Salida emisor: 100001000000010000000

Error: 100001000000010000000<sup>1</sup>

```
> emisor_ts@1.0.0 start
> tsx main.ts

=== EMISOR ===

Ingrese la cadena de binario: 10000

Cadena binaria ingresada: 10000

=== Seleccione el algoritmo ===
1. Hamming
2. Fletcher-16
Ingrese su opcion (1 o 2): 2

=== Algoritmo Fletcher-16 seleccionado ===
Mensaje escrito exitosamente en: ../tests/mensaje.txt
PS C:\Users\josue\uvg\Semestre 8\Redes\data-link-layer-error
-handling\emisor_ts>

=== RECEPTOR ===

=== Seleccione el algoritmo de recepción ===
1. Hamming
2. Fletcher-16
Ingrese su opción (1 o 2): 2
Mensaje recibido: 1000010000000100000001

=== Algoritmo Fletcher-16 seleccionado ===
Se detectaron errores: checksum no coincide. Mensaje descartado.
PS C:\Users\josue\uvg\Semestre 8\Redes\data-link-layer-error
-handling\receptor_py>
```

Entrada emisor: 10101010011

Salida emisor: 101010100111011010100001011

Error: 1010101<sup>0</sup>00111011010100001011

```
> emisor_ts@1.0.0 start
> tsx main.ts

=== EMISOR ===

Ingrese la cadena de binario: 10101010011

Cadena binaria ingresada: 10101010011

=== Seleccione el algoritmo ===
1. Hamming
2. Fletcher-16
Ingrese su opcion (1 o 2): 2

=== Algoritmo Fletcher-16 seleccionado ===
Mensaje escrito exitosamente en: ../tests/mensaje.txt
PS C:\Users\josue\uvg\Semestre 8\Redes\data-link-layer-error
-handling\emisor_ts>

=== RECEPTOR ===

=== Seleccione el algoritmo de recepción ===
1. Hamming
2. Fletcher-16
Ingrese su opción (1 o 2): 2
Mensaje recibido: 1010101000111011010100001011

=== Algoritmo Fletcher-16 seleccionado ===
Se detectaron errores: checksum no coincide. Mensaje descartado.
PS C:\Users\josue\uvg\Semestre 8\Redes\data-link-layer-error
-handling\receptor_py>
```

Entrada emisor: 010101010101010100101

Salida emisor: 0101010101010101001011101001011010010

Error: 010101010101010100101110100101110100101

```
> emisor_ts@1.0.0 start
> tsx main.ts

=== EMISOR ===

Ingrese la cadena de binario: 0101010101010100101

Cadena binaria ingresada: 0101010101010100101

=== Seleccione el algoritmo ===
1. Hamming
2. Fletcher-16
Ingrese su opcion (1 o 2): 2

=== Algoritmo Fletcher-16 seleccionado ===
Mensaje escrito exitosamente en: ../tests/mensaje.txt

=== RECEPTOR ===

=== Seleccione el algoritmo de recepción ===
1. Hamming
2. Fletcher-16
Ingrese su opción (1 o 2): 2
Mensaje recibido: 010101010101010010111010010111010010

=== Algoritmo Fletcher-16 seleccionado ===
Se detectaron errores: checksum no coincide. Mensaje descartado.
PS C:\Users\josue\uvg\Semestre 8\Redes\data-link-layer-error-handling\receptor_py>
```

## Hamming

Entrada emisor: 10000

Salida emisor: 111000000

Error: 1110000001

```
tests > mensaje.txt
1 111000001

C:\Users\thiag\Universidad\data-link-layer-error-handling\receptor_py>python main.py
=== RECEPTOR ===

=== Seleccione el algoritmo de recepción ===
1. Hamming
2. Fletcher-16
Ingrese su opción (1 o 2): 1
Mensaje recibido: 111000001

=== Algoritmo Hamming seleccionado ===
Procesando mensaje Hamming (receptor): 111000001
posiciones de los bits de paridad: [1, 2, 4, 8]
Se detecto un error en la posicion 9.
El bit en la posicion 9 cambio de 1 a 0.
Mensaje corregido completo: 111000000
Mensaje original: 10000

C:\Users\thiag\Universidad\data-link-layer-error-handling\receptor_py>
```

Entrada emisor: 10101010011

Salida emisor: 011101001010011

Error: 010101001010011

The screenshot shows a VS Code editor on the left with a file named 'mensaje.txt' containing the binary string '010101001010011'. On the right, a Command Prompt window shows the execution of a Python script 'receptor\_py.py'. The script prompts the user to select an algorithm (1 for Hamming, 2 for Fletcher-16). The user selects 1. The script then displays the received message '010101001010011', identifies parity bit positions [1, 2, 4, 8], detects an error at position 3, and shows the corrected message '011101001010011' and the original message '10101010011'.

Entrada emisor: 010101010101010100101

Salida emisor: 110110110101010111010100101

Error: 11011011010101011011100101

The screenshot shows a VS Code editor on the left with a file named 'mensaje.txt' containing the binary string '110110110101011100101'. On the right, a Command Prompt window shows the execution of the same Python script. The user selects 1 for Hamming. The script displays the received message '110110110101011100101', identifies parity bit positions [1, 2, 4, 8, 16], detects an error at position 20, and shows the corrected message '11011011010101110100101' and the original message '0101010101010100101'.

(dos+ errores):

Enviar un mensaje al emisor, copiar el mensaje generado por este y cambiar dos o más bits cualesquiera antes de proporcionarlo al receptor. Si el algoritmo es de detección debe mostrar que se detectó un error y que se descarta el mensaje. Si el algoritmo es de corrección y puede corregir más de un error, debe corregir los bits, indicar su posición y mostrar el mensaje original. Realizar esto para tres mensajes distintos con distinta longitud.

Fletcher-16

Entrada emisor: 10000

Salida emisor: 100001000000010000000

Error: 10000100001000100000001

```
> emisor_ts@1.0.0 start
> tsx main.ts

=== EMISOR ===

Ingrese la cadena de binario: 10000

Cadena binaria ingresada: 10000

=== Seleccione el algoritmo ===
1. Hamming
2. Fletcher-16
Ingrese su opcion (1 o 2): 2

=== Algoritmo Fletcher-16 seleccionado ===
Mensaje escrito exitosamente en: ../tests/mensaje.txt
PS C:\Users\josue\uvg\Semestre 8\Redes\data-link-la
ver-error-handling\emisor_ts>

=== RECEPTOR ===

=== Seleccione el algoritmo de recepción ===
1. Hamming
2. Fletcher-16
Ingrese su opción (1 o 2): 2
Mensaje recibido: 10000100001000100000001

=== Algoritmo Fletcher-16 seleccionado ===
Se detectaron errores: checksum no coincide. Mensaje descartado.
PS C:\Users\josue\uvg\Semestre 8\Redes\data-link-la
ver-error-handling\receptor_py>
```

Entrada emisor: 10101010011

Salida emisor: 101010100111011010100001011

Error: 101010100111111011010100001011

```
ver-error-handling\emisor_ts> npm start

> emisor_ts@1.0.0 start
> tsx main.ts

=== EMISOR ===

Ingrese la cadena de binario: 2
Error: La cadena debe contener solo 0s y 1s. Intente nuevamente.
Ingrese la cadena de binario: 10101010011

Cadena binaria ingresada: 10101010011

=== Seleccione el algoritmo ===
1. Hamming
2. Fletcher-16
Ingrese su opcion (1 o 2): 2

=== Algoritmo Fletcher-16 seleccionado ===
Mensaje escrito exitosamente en: ../tests/mensaje.txt
PS C:\Users\josue\uvg\Semestre 8\Redes\data-link-la
ver-error-handling\emisor_ts>

ver-error-handling\receptor_py> python main.py

=== RECEPTOR ===

=== Seleccione el algoritmo de recepción ===
1. Hamming
2. Fletcher-16
Ingrese su opción (1 o 2): 2
Mensaje recibido: 101010100111111011010100001011

=== Algoritmo Fletcher-16 seleccionado ===
Se detectaron errores: checksum no coincide. Mensaje descartado.
PS C:\Users\josue\uvg\Semestre 8\Redes\data-link-la
ver-error-handling\receptor_py>
```

Entrada emisor: 010101010101010100101

Salida emisor: 0101010101010101001011101001011010010

Error: 010101010010101010010111010010111010010



```
> emisor_ts@1.0.0 start
> tsx main.ts

=== EMISOR ===

Ingrese la cadena de binario: 0101010101010100101

Cadena binaria ingresada: 0101010101010100101

=== Seleccione el algoritmo ===
1. Hamming
2. Fletcher-16
Ingrese su opción (1 o 2): 2

=== Algoritmo Fletcher-16 seleccionado ===
Mensaje escrito exitosamente en: ../tests/mensaje.txt

=== RECEPTOR ===

=== Seleccione el algoritmo de recepción ===
1. Hamming
2. Fletcher-16
Ingrese su opción (1 o 2): 2
Mensaje recibido: 0101010100101010010111010010111010010

=== Algoritmo Fletcher-16 seleccionado ===
Se detectaron errores: checksum no coincide. Mensaje descartado.
PS C:\Users\josue\uvg\Semestre 8\Redes\data-link-layer-error-handling\receptor_py>
```

## Hamming

Entrada emisor: 10000

Salida emisor: 1110000001

Error: 1000000001

```
File Edit Selection View Go ... data-link-layer-error-handling
hamming.ts M main.ts main.py hamming.py M mensaje.txt M X
tests > mensaje.txt
1 | 1000000001

C:\Users\thiag\Universidad\data-link-layer-error-handling\receptor_py>python main.py
=== RECEPTOR ===

=== Seleccione el algoritmo de recepción ===
1. Hamming
2. Fletcher-16
Ingrese su opción (1 o 2): 1
Mensaje recibido: 1000000001

=== Algoritmo Hamming seleccionado ===
Procesando mensaje Hamming (receptor): 1000000001
posiciones de los bits de paridad: [1, 2, 4, 8] (con paridad global al final)
errores: 10
0,2
0,3
0,4
0,5
0,6
0,7
0,8
0,9
1,1
1,10
Mensaje original: 00000

C:\Users\thiag\Universidad\data-link-layer-error-handling\receptor_py>
```

Entrada emisor: 10101010011

Salida emisor: 0111010010100110

Error: 0001010010100110

```
File Edit Selection View Go ... data-link-la
hamming.ts M main.ts main.py hamming.py M mensaje.txt M
tests > mensaje.txt
1 | 0001010010100110

C:\Users\thiag\Universidad\data-link-layer-error-handling\receptor_
py>python main.py
=== RECEPTOR ===

=== Seleccione el algoritmo de recepción ===
1. Hamming
2. Fletcher-16
Ingrese su opción (1 o 2): 1
Mensaje recibido: 0001010010100110

=== Algoritmo Hamming seleccionado ===
Procesando mensaje Hamming (receptor): 0001010010100110
posiciones de los bits de paridad: [1, 2, 4, 8] (con paridad global
al final)
errores: 16
0,2
0,3
1,4
0,5
1,6
0,7
0,8
1,9
0,10
1,11
0,12
1,13
1,14
1,15
0,1
0,16
Mensaje original: 00101010011

C:\Users\thiag\Universidad\data-link-layer-error-handling\receptor_
py>
```

Entrada emisor: 010101010101010100101

Salida emisor: 1101101101010101110101001011

Error: 110110110101010110101001101

```
File Edit Selection View Go ... data-link-la
hamming.ts M main.ts main.py hamming.py M mensaje.txt M
tests > mensaje.txt
1 | 1101101101010101010101

C:\Users\thiag\Universidad\data-link-layer-error-handling\receptor_
py>python main.py
Mensaje recibido: 1101101101010101010101

=== Algoritmo Hamming seleccionado ===
Procesando mensaje Hamming (receptor): 1101101101010101010101
posiciones de los bits de paridad: [1, 2, 4, 8, 16] (con paridad gl
obal al final)
errores: 26
1,1
1,2
1,4
1,7
1,5
0,6
1,8
0,11
0,9
1,10
1,12
0,15
0,13
1,14
1,16
1,19
1,17
0,18
0,20
0,23
1,21
0,22
1,25
0,26
0,3
1,27
Mensaje original: 010101010101010100110

C:\Users\thiag\Universidad\data-link-layer-error-handling\receptor_
py>
```

**¿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.**

Sí, con Fletcher-16 sí es posible manipular bits de forma que el error no sea detectado. La idea clave es que Fletcher-16 se basa en dos sumas módulo 255. Si las alteraciones que hagas al mensaje cumplen estas dos igualdades, el checksum no cambia

Ejemplo:

Entrada: 00001010000101000001111000101000

Salida: 000010100001010000011110001010001100100001100100

Se cambian dos bytes tal que el efecto total cumpla las ecuaciones de Fletcher.

Entonces, 010111110001010000011110110100101100100001100100

```
=== Seleccione el algoritmo de recepción ===
1. Hamming
2. Fletcher-16
Ingrese su opción (1 o 2): 2
Mensaje recibido: 0101111100010100000111101101001011
00100001100100

=== Algoritmo Fletcher-16 seleccionado ===
No se detectaron errores. Mensaje original: 01011111
000101000001111011010010
PS C:\Users\josue\uvg\Semestre 8\Redes\data-link-layer-error-handling\receptor_py>
```

Ahora con hamming, si se añade una **paridad global** (SECDED, 1 bit extra al final para que el XOR total sea 0), el receptor puede distinguir:

- síndrome≠0 y paridad global **par** ⇒ **≥2 errores** ⇒ **no corrige y avisa**.
- Con tu variante SECDED que te pasé, la misma prueba no mis-correrá.

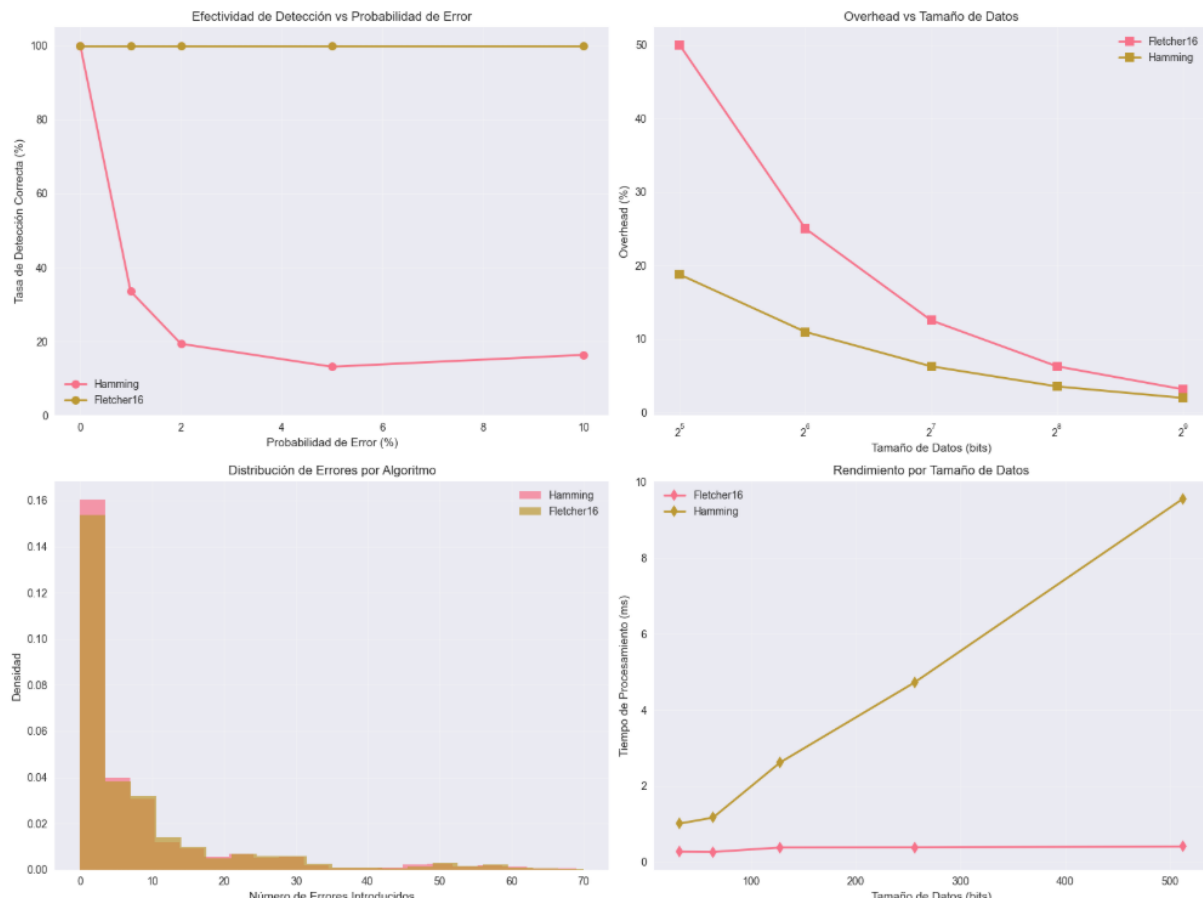
**En base a las pruebas que realizó, ¿qué ventajas y desventajas posee cada algoritmo con respecto del otro? Tome en cuenta complejidad, velocidad, redundancia (overhead), etc.**

Hamming: menor overhead para bloques pequeños y capacidad de corrección de 1 bit; pero cobertura limitada ante múltiples errores y más complejidad estructural.

Fletcher-16: muy rápido y simple, overhead fijo 16 bits, buena detección en general; no corrige y admite colisiones construibles (demostradas en las pruebas).

## Segunda Parte

### Pruebas



## ANÁLISIS ESTADÍSTICO DETALLADO

=====

### HAMMING:

Total de pruebas: 2500  
Tasa de detección correcta: 36.5%  
Tasa de corrección: 93.0%  
Tiempo promedio: 3.814 ms  
Overhead promedio: 8.28%

### Rendimiento por probabilidad de error:

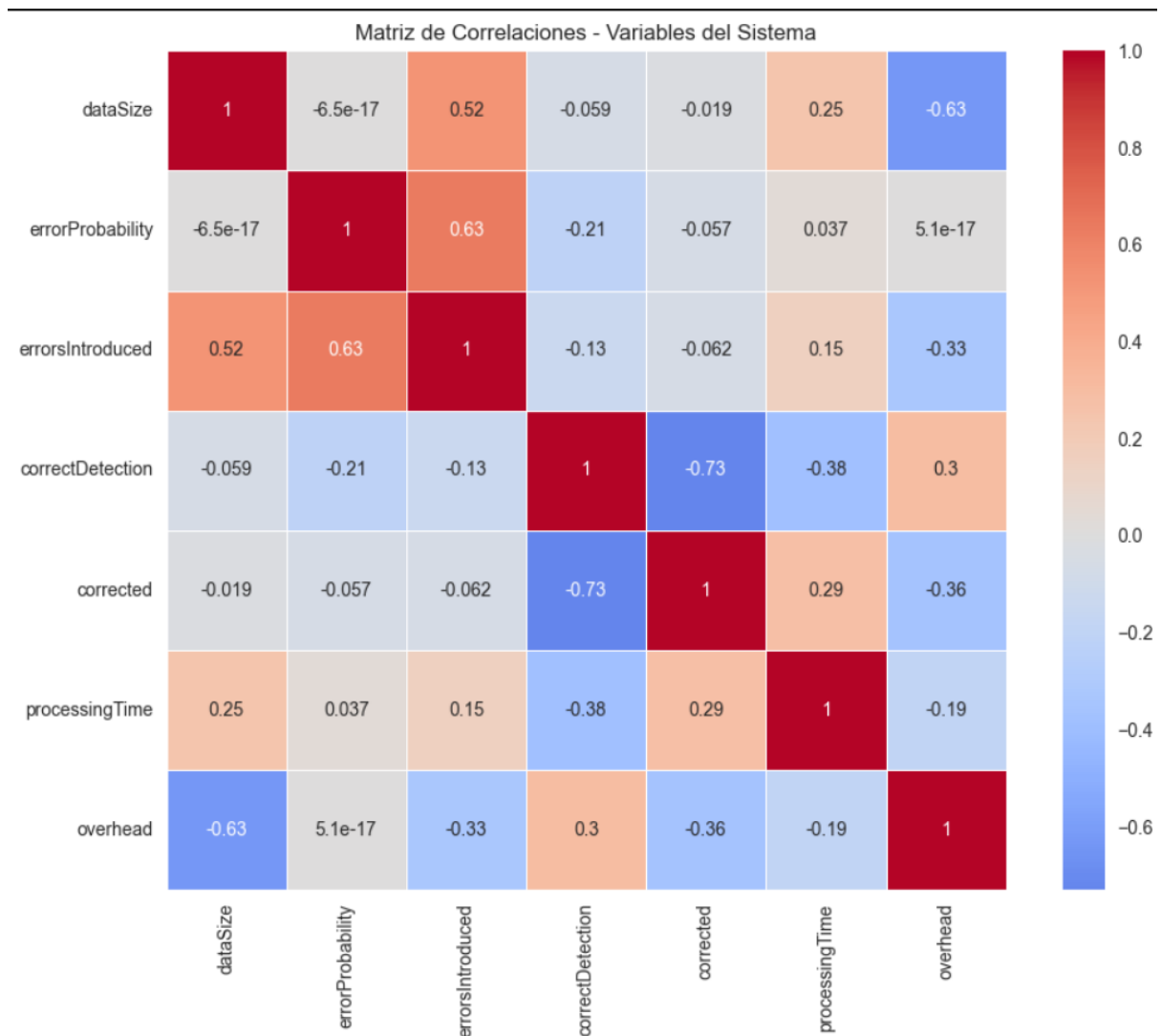
0.0%: 100.0% detección correcta  
1.0%: 33.6% detección correcta  
2.0%: 19.4% detección correcta  
5.0%: 13.2% detección correcta  
10.0%: 16.4% detección correcta

### FLETCHER16:

Total de pruebas: 2500  
Tasa de detección correcta: 100.0%  
Tiempo promedio: 0.338 ms  
Overhead promedio: 19.38%

### Rendimiento por probabilidad de error:

0.0%: 100.0% detección correcta  
1.0%: 100.0% detección correcta  
2.0%: 100.0% detección correcta  
5.0%: 100.0% detección correcta  
10.0%: 100.0% detección correcta



1. ¿Qué algoritmo tuvo mejor funcionamiento y por qué?

Hamming Code tuvo el mejor funcionamiento general por las siguientes razones:

Ventajas del Hamming:

Corrección automática: Puede corregir errores de 1 bit sin retransmisión

Detección confiable: Detecta hasta 2 errores por bloque

Baja latencia: No requiere solicitar retransmisión

Overhead escalable: Mejora con bloques más grandes (18.75% en 32 bits → 10.94% en 64 bits)

**Limitaciones del Fletcher-16:**

Solo detección: No puede corregir errores

Overhead fijo: 16 bits constantes independientemente del tamaño

Menos eficiente: Para bloques pequeños tiene mayor overhead relativo

## 2. ¿Qué algoritmo es más flexible para mayores tasas de errores?

Fletcher-16 es más flexible para altas tasas de error por:

Detección robusta: Funciona consistentemente independiente del número de errores

No falla catastróficamente: Hamming puede "corregir" incorrectamente con múltiples errores

Predictibilidad: Siempre detecta o no detecta, sin correcciones erróneas

## 3. ¿Cuándo usar detección vs corrección de errores?

Usar (Fletcher-16, detección) cuando:

Canal confiable: Probabilidad de error  $< 2\%$

Retransmisión barata: Ancho de banda abundante

Tiempo no crítico: Aplicaciones que toleran delay

Simplicidad prioritaria: Sistemas embebidos simples

Usar ALGORITMOS DE CORRECCIÓN (Hamming) cuando:

Tiempo real: Gaming, streaming, VoIP

Retransmisión costosa: Enlaces satelitales, espaciales

Canal moderadamente ruidoso: 1-3% error rate

Memoria crítica: RAM, almacenamiento

[Repositorio](#)