

Laboratorio 2

Redes

José Luis Gramajo Moraga, Carné 22907
Angel Andres Herrarte Lorenzana, Carné 22873

24 de julio de 2025 Guatemala, Guatemala

Corrección de errores – Algoritmo de Hamming

El Código de Hamming utiliza bits de paridad posicionados estratégicamente en potencias de 2 (posiciones 1, 2, 4, 8, 16...) para crear un "síndrome" que indica tanto la presencia como la ubicación exacta de un error de un bit. La fórmula que rige la implementación es:

$$m + r + 1 \leq 2^r$$

Donde:

- m = número de bits de datos
- r = número de bits de paridad necesarios

Implementación

El sistema se implementó usando dos componentes separados en lenguajes diferentes:

- **Emisor:** JavaScript (Node.js) - Genera códigos de Hamming
- **Receptor:** Python - Detecta y corrige errores

El emisor sigue estos pasos:

1. **Cálculo de bits de paridad necesarios:** Usando la fórmula $m + r + 1 \leq 2^r$
2. **Posicionamiento de datos:** Los bits de datos se colocan en posiciones que no son potencias de 2
3. **Cálculo de paridades:** Cada bit de paridad cubre un patrón específico de posiciones
4. **Construcción de trama final:** Se combinan datos y bits de paridad

El receptor implementa la detección y corrección:

1. **Cálculo del síndrome:** Verificación de cada bit de paridad
2. **Análisis del síndrome:** Si síndrome = 0, no hay errores; si $\neq 0$, indica posición del error
3. **Corrección automática:** Inversión del bit en la posición indicada por el síndrome
4. **Extracción de datos:** Recuperación de los bits de datos originales

Escenarios de prueba

Se utilizaron tres mensajes con diferentes longitudes para evaluar la escalabilidad:

Mensaje	Longitud	Trama Codificada	Overhead
1101	4 bits	1010101 (7 bits)	75%

10110	5 bits	011001100 (9 bits)	80%
111010011	9 bits	0010110110011 (13 bits)	44%

- Transmisión Sin Errores

Mensaje 1: "1101"

```

Trama enviada: 1010101
Resultado: ✓ No se detectaron errores
Datos recuperados: 1101
Síndrome: 0

```

Mensaje 2: "10110"

```

Trama enviada: 011001100
Resultado: ✓ No se detectaron errores
Datos recuperados: 10110
Síndrome: 0

```

Mensaje 3: "111010011"

```

Trama enviada: 0010110110011
Resultado: ✓ No se detectaron errores
Datos recuperados: 111010011
Síndrome: 0

```

Todas las tramas sin errores fueron procesadas correctamente, confirmando la implementación correcta del algoritmo.

- Transmisión Con 1 Error

Mensaje 1: "1101"

```
Trama original: 1010101
Trama con error: 1000101 (error en posición 3)
Síndrome calculado: 3
Resultado: ✓ Error detectado y corregido en posición 3
Trama corregida: 1010101
Datos recuperados: 1101
```

Mensaje 2: "10110"

```
Trama original: 011001100
Trama con error: 010001100 (error en posición 3)
Síndrome calculado: 3
Resultado: ✓ Error detectado y corregido en posición 3
Trama corregida: 011001100
Datos recuperados: 10110
```

Mensaje 3: "111010011"

```
Trama original: 0010110110011
Trama con error: 0000110110011 (error en posición 3)
Síndrome calculado: 3
Resultado: ✓ Error detectado y corregido en posición 3
Trama corregida: 0010110110011
Datos recuperados: 111010011
```

El algoritmo detectó y corrigió todos los errores de un bit, demostrando su eficacia en este escenario.

- Transmisión Con 2 + Errores, con errores en posiciones 2 y 5 para cada mensaje

Mensaje 1: "1101"

```
Trama original: 1010101
Trama con 2 errores: 1110001 (errores en posiciones 2, 5)
Síndrome calculado: 7
Resultado: ✗ Error "detectado" en posición 7 (incorrecta)
Trama "corregida": 1110000
Datos "recuperados": 1000 (incorrecto)
```

Mensaje 2: "10110"

```
Trama original: 011001100
Trama con 2 errores: 001011100 (errores en posiciones 2, 5)
Síndrome calculado: 7
Resultado: ✗ Error "detectado" en posición 7 (incorrecta)
Trama "corregida": 001011000
Datos "recuperados": 11100 (incorrecto)
```

Mensaje 3: "111010011"

```
Trama original: 0010110110011
Trama con 2 errores: 0110010110011 (errores en posiciones 2, 5)
Síndrome calculado: 7
Resultado: ✗ Error "detectado" en posición 7 (incorrecta)
Trama "corregida": 0110011110011
Datos "recuperados": 101110011 (incorrecto)
```

Mensaje 1: "1101"

```
Trama original: 1010101
Trama con 2 errores: 1110001 (errores en posiciones 2, 5)
Síndrome calculado: 7
Resultado: × Error "detectado" en posición 7 (incorrecta)
Trama "corregida": 1110000
Datos "recuperados": 1000 (incorrecto)
```

Mensaje 2: "10110"

```
Trama original: 011001100
Trama con 2 errores: 001011100 (errores en posiciones 2, 5)
Síndrome calculado: 7
Resultado: × Error "detectado" en posición 7 (incorrecta)
Trama "corregida": 001011000
Datos "recuperados": 11100 (incorrecto)
```

Mensaje 3: "111010011"

```
Trama original: 0010110110011
Trama con 2 errores: 0110010110011 (errores en posiciones 2, 5)
Síndrome calculado: 7
Resultado: × Error "detectado" en posición 7 (incorrecta)
Trama "corregida": 0110011110011
Datos "recuperados": 101110011 (incorrecto)
```

Con errores múltiples, el algoritmo detecta la presencia de errores, pero los localiza incorrectamente, resultando en "correcciones" que empeoran la situación. Esta es una limitación fundamental del Código de Hamming simple.

- ¿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, demuéstrelo con su implementación.

El algoritmo puede fallar en detectar errores cuando múltiples errores se combinan de manera que el síndrome resultante sea cero. Esto ocurre cuando los errores siguen patrones específicos que se "cancelan" en el cálculo de paridad.

Caso No Detectable Identificado

- Mensaje: "1101"
- Trama correcta: 1010101

Patrón de errores no detectable

- Errores en posiciones: [1, 3, 5, 7]
- Trama con errores: 0000000
- Síndrome calculado: 0
- Resultado: Error no detectado

Explicación Técnica

Este patrón de errores afecta todas las posiciones cubiertas por el bit de paridad P1, pero de manera que las verificaciones de paridad siguen dando resultados válidos:

- **P1** (posiciones 1,3,5,7): $\text{XOR}(0,0,0,0) = 0$ (Correcto)
- **P2** (posiciones 2,3,6,7): $\text{XOR}(1,0,1,0) = 0$ (Correcto)
- **P4** (posiciones 4,5,6,7): $\text{XOR}(0,0,1,0) = 1...$ (Incorrecto)

Detección de errores – Fletcher Checksum

1. Escenarios de pruebas

Para cada combinación de mensaje y tamaño de bloque (8, 16 y 32 bits) se realizaron tres casos:

1. Sin errores
2. Un bit alterado
3. Dos bits alterados

1.1. Caso: mensaje = b"A" ; block_size = 8 bits

Paso	Valor / Descripción
Mensaje original	b"A" (0x41)
Emisor: cálculo	– $\text{sum1} = 0x41 (65) > 65 \bmod 255 \rightarrow 0x41$ – $\text{sum2} = 0 + 0x41 \rightarrow 0x41$
Trama emitida	$\text{data} \parallel \text{sum1} \parallel \text{sum2} \rightarrow [0x41] \parallel [0x41] \parallel [0x41] \rightarrow 0x41 \ 0x41 \ 0x41$

1 bit alterado	Invierte bit 0 (LSB) del primer byte: $0x41 \rightarrow 0x40$
Trama corrupta: $0x40 \ 0x41 \ 0x41$	
Receptor: verificación	– Cálculo sobre $\text{data} = [0x40]$: $\text{sum1} = \text{sum2} = 0x40$. – Checksum esperado: $[0x40 \parallel 0x40]$, pero llegó $[0x41 \parallel 0x41]$.
→ Error detectado; trama descartada.	

2 bits alterados	Invierte bits 0 y 1 del primer byte: $0x41 \rightarrow 0x40 \rightarrow 0x42$
Trama corrupta: $0x42 \ 0x41 \ 0x41$	
→ Error detectado (mismatched checksum).	
(Sin errores)	Receptor recibe $0x41 \ 0x41 \ 0x41$, recalcula $\text{sum1}=\text{sum2}=0x41$ y retorna (True, b"A").

En todos los 27 casos (3 mensajes × 3 tamaños × 3 escenarios) el algoritmo:

- Acepta correctamente las tramas sin errores.
- Detecta 1 bit alterado.

- Detecta ≥ 2 bits alterados.

2. Análisis de colisiones y fiabilidad

¿Puede un atacante modificar bits sin ser detectado?

- Teóricamente sí: existen pares de mensajes distintos que producen el mismo par de sumas (sum1,sum2), sobre todo en bloques pequeños (8 bits).
- Demostración simple (Fletcher-8):

Conclusión:

- Fletcher es fuerte frente a un solo bit corrupto (detecta siempre).
- Sin embargo, no garantiza detección absoluta de errores múltiples o manipulaciones maliciosas que exploten colisiones de checksum, especialmente en bloques de 8 bits.

Análisis comparativo

Ventajas del Código de Hamming

1. Corrección automática: Único algoritmo evaluado capaz de corregir errores sin retransmisión
2. Precisión en localización: Indica la posición exacta del error de un bit
3. Determinismo: Siempre produce el mismo resultado para la misma entrada
4. Eficiencia para errores únicos: 100% de efectividad en corrección de errores de un bit

Desventajas del Código de Hamming

1. Alto overhead: Para mensajes cortos, puede requerir hasta 75% de bits adicionales
2. Limitación a errores únicos: Falla con errores múltiples
3. Complejidad computacional: Requiere más procesamiento que algoritmos de detección simple
4. Casos no detectables: Existen patrones específicos que resultan en falsos negativos

Aspecto	Hamming	Fletcher
Función principal	Corrección de errores	Detección de errores
Capacidad de corrección	1 bit perfecto	No aplica
Detección de errores múltiples	Limitada (puede fallar)	Excelente
Overhead	Variable (44-80%)	Fijo (2 bytes)
Velocidad de procesamiento	Media	Rápida
Complejidad de implementación	Media-Alta	Baja
Casos de uso ideales	Canales con errores únicos frecuentes	Detección general en redes

Código Fuente

<https://github.com/Abysswalkr/Lab2-Deteccion-Correccion-Errores>