Laboratorio 2 - Parte 2: Algoritmos de corrección
Redes

Hugo Rivas - 22500
Alexis Mesias - 22562

Pruebas:
Hamming:

```
PS C:\Users\Eduar\OneDrive\Documentos\REDES\Lab2.2-Redes\Emisor> python Emisor.py
Ingrese el mensaje a enviar: Hola
Ingrese la tasa de error (ej. 1/100 para 1 error cada 100 bits): 1/100
Seleccione el algoritmo de detección de errores:
1) Paridad par
2) Hamming
3) CRC-32
Ingrese el número de la opción: 2
Mensaje codificado en binario: 01001000011011110110110001100001
Mensaje con hamming: 11001001100001110111101101100010100001
Mensaje final con ruido: 11001001100001110111101101100010100001
Payload a enviar: hamming|11001001100001110111101101100010100001
Mensaje enviado exitosamente.
```

```
PS C:\Users\Eduar\OneDrive\Documentos\REDES\Lab2.2-Redes>  c:; cd 'c:\Users\Eduar\OneDrive\Documentos\REDES\Lab2.2-Redes'; & 'C:\Program Files\Java\jdk-21\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Eduar\AppData\Roaming\Code\User\workspaceStorage\067ff4012d6ccad3e7ae211344a6a2e6\redhat.java\jdt_ws\Lab2.2-Redes_4404a637\bin' 'Receptor'
Mensaje recibido: hamming|11001001100001110111101101100010100001
```

CRC:

```
PS C:\Users\Eduar\OneDrive\Documentos\REDES\Lab2.2-Redes\Emisor> python Emisor.py
Ingrese el mensaje a enviar: HOLAAAA
Ingrese la tasa de error (ej. 1/100 para 1 error cada 100 bits): 50/100
Seleccione el algoritmo de detección de errores:
1) Paridad par
2) Hamming
3) CRC-32
Ingrese el número de la opción: 3
Mensaje codificado en binario: 01001000010011110100110001000001010000010100000101000001
Mensaje con crc32: 01001000010011110100110001000001010000010100000101000001001100111100110000111110011111100
Mensaje final con ruido: 00011000111011100000100000000101011110011010011110100011111010101100110000010011111011000
Payload a enviar: crc32|00011000111011100000100000000101011110011010011110100011111010101100110000010011111011000
Mensaje enviado exitosamente.
```

```
PS C:\Users\Eduar\OneDrive\Documentos\REDES\Lab2.2-Redes>  c:; cd 'c:\Users\Eduar\OneDrive\Documentos\REDES\Lab2.2-Redes'; & 'C:\Program Files\Java\jdk-21\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Eduar\AppData\Roaming\Code\User\workspaceStorage\067ff4012d6ccad3e7ae211344a6a2e6\redhat.java\jdt_ws\Lab2.2-Redes_4404a637\bin' 'Receptor'
Mensaje recibido: crc32|00011000111011100000100000000101011110011010011110100011111010101100110000010011111011000
Error detectado en la transmisión. El mensaje podría estar corrupto.
```

Preguntas:
¿Qué algoritmo tuvo un mejor funcionamiento?

El algoritmo que tuvo un mejor funcionamiento fue Hamming, ya que fue capaz de detectar y corregir un error en la transmisión, permitiendo recuperar el mensaje original sin necesidad de reenvío.

Esto se observa en la primera imagen, donde a pesar de la introducción de ruido, el receptor logró reconstruir el mensaje correctamente.

En cambio, el algoritmo CRC-32, aunque también detectó el error, no tiene capacidad de corrección, por lo que únicamente notificó que el mensaje estaba corrupto.

¿Qué algoritmo es más flexible para aceptar mayores tasas de errores?

El algoritmo más flexible ante mayores tasas de errores es CRC-32, ya que:

- Es muy eficaz para detectar múltiples errores en largas secuencias de bits.
- Tiene una alta probabilidad de detectar errores complejos como errores en ráfaga.

Sin embargo, su limitación es que no puede corregir errores, solo detectarlos.

Hamming, por su parte:

- Tiene capacidad de corrección de un solo bit por bloque.
- Si se supera este límite, la corrección falla o incluso puede corromper el mensaje.

Por tanto, para entornos con mucho ruido (como en la imagen con tasa 50/100), CRC es más robusto para detección, pero no para recuperación automática.

¿Cuándo es mejor utilizar un algoritmo de detección de errores en lugar de uno de corrección de errores?

Es mejor utilizar un algoritmo solo de detección (como CRC o paridad) cuando:

1. Se dispone de un canal confiable de retroalimentación:
    - Por ejemplo, si el receptor puede pedir una retransmisión automática del mensaje (como en protocolos como TCP).
2. Se prioriza la velocidad y eficiencia, y los errores son poco frecuentes.
3. Se transmite gran cantidad de datos y la corrección automática sería demasiado costosa o poco eficiente.

En cambio, los algoritmos de corrección como Hamming son útiles cuando:

- No es posible pedir retransmisión (por ejemplo, en comunicación satelital o transmisión de audio en tiempo real).
- Se requiere autonomía en la recuperación del mensaje.