

LABORATORIO

NO. 2

Esquemas de detección y corrección: Parte 1 y 2

Marco Ramirez 21032 | Josué Morales 21116

Tabla de contenidos

Corrección de errores.....2

Algoritmo de Hamming2

 Pruebas 2

Detección de errores.....4

Fletcher checksum.....4

 Pruebas 4

Discusión.....6

Conclusiones7

Bibliografía18

Parte 1

Corrección de errores

Algoritmo de Hamming

Es una técnica de corrección de errores diseñada para asegurar la exactitud de los datos durante su transmisión o almacenamiento. Este código identifica y corrige errores que pueden surgir en el proceso de enviar o almacenar información desde el remitente hasta el receptor. (GeeksforGeeks, 2024)

Pruebas

(GeekforGeeks, Hamming code Implementation in Java , 2020)

- 0101011

```
> java codigo.java                                     17:55:21 [0/7]
Ingrese el mensaje binario
0101011
0 1 0 1 0 1 1
Parity bits
1
1
0
0
Ingrese la cantidad de errores a agregar
0
No hay errores
Data del emisor:
1101010

~/Library/CloudStorage/OneDrive-UVG/Documentos/Semestre_8/Redes/Redes_2024 Lab-2_1*
>
[3] 0:[tmux]*                                           "Marcos-MacBook-Pro-6." 17:57 19-Jul-24
```

- 000110

```
> java codigo.java
Ingrese el mensaje binario
000110
0 0 0 1 1 0
Parity bits
0
1
1
1
Ingrese la cantidad de errores a agregar
0
No hay errores
Data del emisor:
011000

~/Library/CloudStorage/OneDrive-UVG/Documentos/Semestre_8/Redes/Redes_2024 Lab-2_1*
>
[3] 0:zsh* "Marcos-MacBook-Pro-6." 18:00 19-Jul-24
```

- 1000111

```
> java codigo.java
Ingrese el mensaje binario
1000111
1 0 0 0 1 1 1
Parity bits
1
1
0
1
Ingrese la cantidad de errores a agregar
0
No hay errores
Data del emisor:
1110001

~/Library/CloudStorage/OneDrive-UVG/Documentos/Semestre_8/Redes/Redes_2024 Lab-2_1* 12s
>
[3] 0:zsh* "Marcos-MacBook-Pro-6." 18:01 19-Jul-24
```

Detección de errores

Fletcher checksum

Es un método de detección de errores utilizado por los protocolos de capa superior y se considera más fiable que el LRC, el VRC y el CRC. Este método emplea un Generador de Checksum en el lado del emisor y un Verificador de Checksum en el lado del receptor.

En el lado del emisor, el generador de checksum divide los datos en subunidades de n bits (generalmente de 16 bits), las suma usando el método de complemento a uno, y luego complementa el resultado. Este checksum complementado se añade al final de los datos originales y se envía al receptor. (GeekforGeeks, 2024)

Pruebas

(yadav, 2020)

- 1010001

```
> python3 detectionFletcher.py
Ingrese la trama de datos en formato binario: 1010001
Ingrese el tamaño del bloque (8, 16 o 32): 8
Checksum calculado para datos originales: 1010001010100010
Trama de datos recibidos sin errores: 10100010
Checksum recibido: 1010001010100010
El checksum es válido.

~/OneDrive - UVG/Documentos/Semestre_8/Redes/Redes_2024 Lab-2_1* 48s
> █

[2] 0:zsh* "Marcos-MacBook-Pro-6." 17:29 19-Jul-24
```

- 1101001

```
> python3 detectionFletcher.py
Ingrese la trama de datos en formato binario: 1101001
Ingrese el tamaño del bloque (8, 16 o 32): 8
Checksum calculado para datos originales: 1101001011010010
Trama de datos recibidos sin errores: 11010010
Checksum recibido: 1101001011010010
El checksum es válido.

~/OneDrive - UVG/Documentos/Semestre_8/Redes/Redes_2024 Lab-2_1* 6s
> █

[2] 0:zsh* "Marcos-MacBook-Pro-6." 17:34 19-Jul-24
```

- 0011101

```
> python3 detectionFletcher.py
Ingrese la trama de datos en formato binario: 0011101
Ingrese el tamaño del bloque (8, 16 o 32): 8
Checksum calculado para datos originales: 11101000111010
Trama de datos recibidos sin errores: 00111010
Checksum recibido: 11101000111010
El checksum es válido.

~/OneDrive - UVG/Documentos/Semestre_8/Redes/Redes_2024 Lab-2_1*
> █

[2] 0:zsh* "Marcos-MacBook-Pro-6." 17:36 19-Jul-24
```

Discusión

El análisis de los algoritmos de corrección y detección de errores, específicamente el código de Hamming y el checksum de Fletcher, tiene algunas características clave y limitaciones en la práctica.

Para el **algoritmo de Hamming** implementado en Java, observamos que la detección y corrección de errores están limitadas a un solo error por trama. Aunque el algoritmo es eficaz para corregir errores simples y detectar errores en una sola posición, no es capaz de identificar más de un error en la trama. Esto se debe a que el algoritmo se ajusta a una estructura específica que solo detecta el primer error. Una posible mejora sería la incorporación de recursividad en la función de corrección, lo que permitiría detectar y corregir múltiples errores. Sin embargo, se optó por mantener la estructura original para no desviarse de los estándares establecidos para el algoritmo.

Por otro lado, en el caso del **checksum de Fletcher** desarrollado en Python, se diseñó una función que generaba tramas de datos aleatorios para simular errores y verificar la capacidad del algoritmo para detectarlos. En los experimentos realizados con datos correctos, no se detectaron errores, pero el algoritmo mostró que podía identificar el primer error en las tramas alteradas. Similar al código de Hamming, el checksum de Fletcher también tiene la limitación de detectar solo el primer error.

Conclusiones

- El código de Hamming y el checksum de Fletcher solo detectan y corrigen el primer error en una trama.
- Ambos métodos son eficaces para la corrección y detección de errores simples, pero limitados en escenarios con múltiples errores.
- El código de Hamming sigue estándares estrictos, lo que restringe su capacidad para detectar errores adicionales más allá del primero.
- El checksum de Fletcher, aunque flexible, también muestra limitaciones similares en la corrección de errores múltiples.

Parte 2

En esta parte del laboratorio, se desarrolló un sistema de comunicación cliente-servidor que emplea los algoritmos de Hamming y Fletcher para la detección y corrección de errores en mensajes transmitidos. El sistema consta de un cliente en Java y un servidor en Python, y tiene los siguientes objetivos:

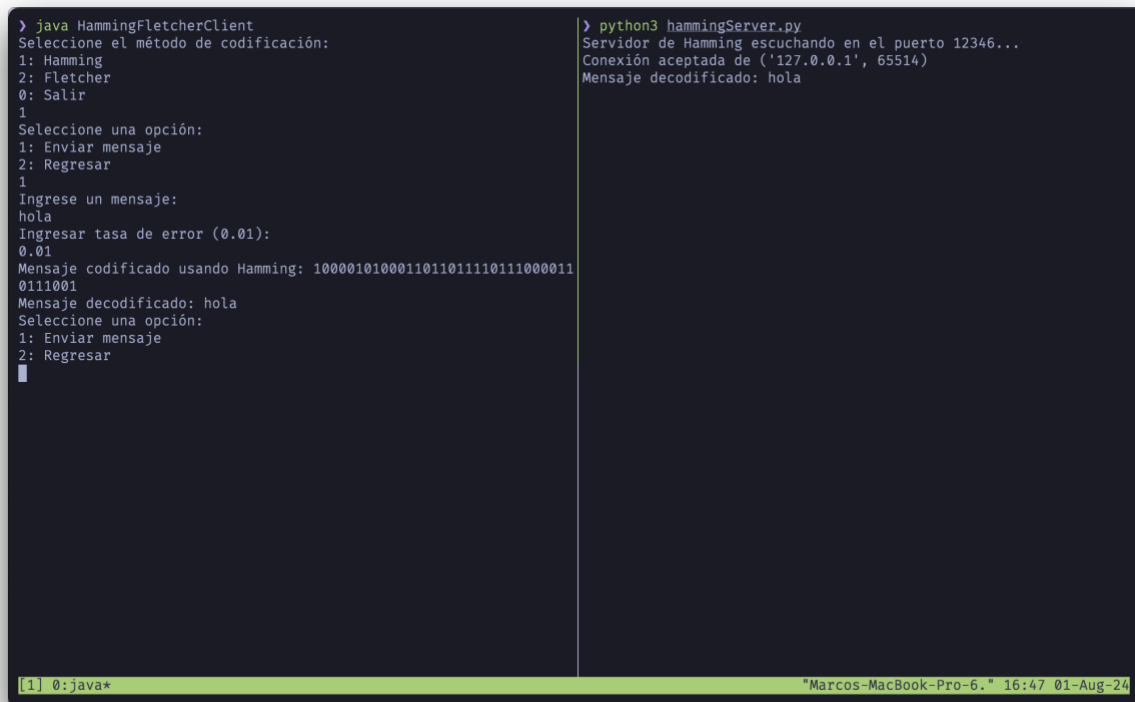
1. **Codificación de Mensajes:** El cliente en Java codifica los mensajes
2. **Introducción de Ruido:** El cliente aplica un nivel configurable de ruido a los mensajes codificados. Este ruido simula errores que pueden ocurrir durante la transmisión de datos a través de la red.
3. **Transmisión y Decodificación:** El mensaje ruidoso se envía al servidor correspondiente, que está configurado para manejar uno de los dos métodos de codificación. El servidor recibe el mensaje, lo decodifica y verifica utilizando el algoritmo correspondiente (Hamming o Fletcher), corrige los errores detectados y devuelve el mensaje decodificado al cliente.

Pruebas

Algoritmo de Hamming

Prueba No.1

- Mensaje: Hola
- Cadena binaria: 10000101000110110111101110000110111001
- Probabilidad de error: 0.01



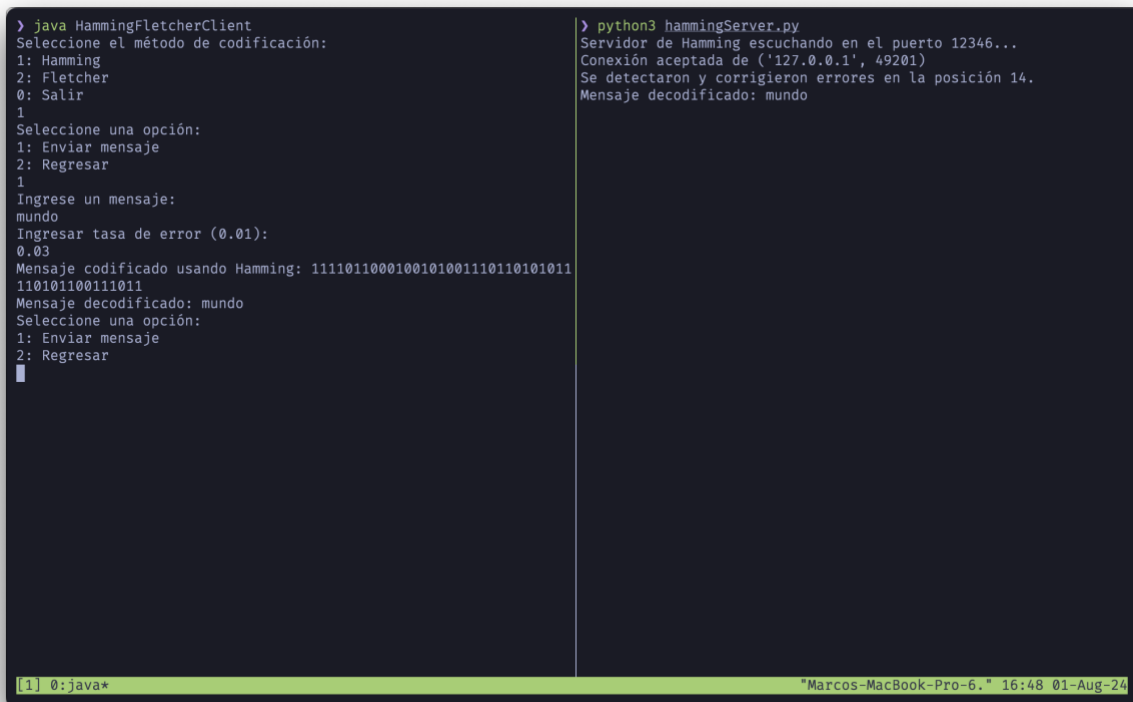
```
> java HammingFletcherClient
Seleccione el método de codificación:
1: Hamming
2: Fletcher
0: Salir
1
Seleccione una opción:
1: Enviar mensaje
2: Regresar
1
Ingrese un mensaje:
hola
Ingrese tasa de error (0.01):
0.01
Mensaje codificado usando Hamming: 1000010100011011011110111000011
0111001
Mensaje decodificado: hola
Seleccione una opción:
1: Enviar mensaje
2: Regresar
1
```

```
> python3 hammingServer.py
Servidor de Hamming escuchando en el puerto 12346...
Conexión aceptada de ('127.0.0.1', 65514)
Mensaje decodificado: hola
```

[1] 0:java* "Marcos-MacBook-Pro-6." 16:47 01-Aug-24

Prueba No.2

- Mensaje: mundo
- Cadena binaria: 1111011000100101001110110101011110101100111011
- Probabilidad de error: 0.03



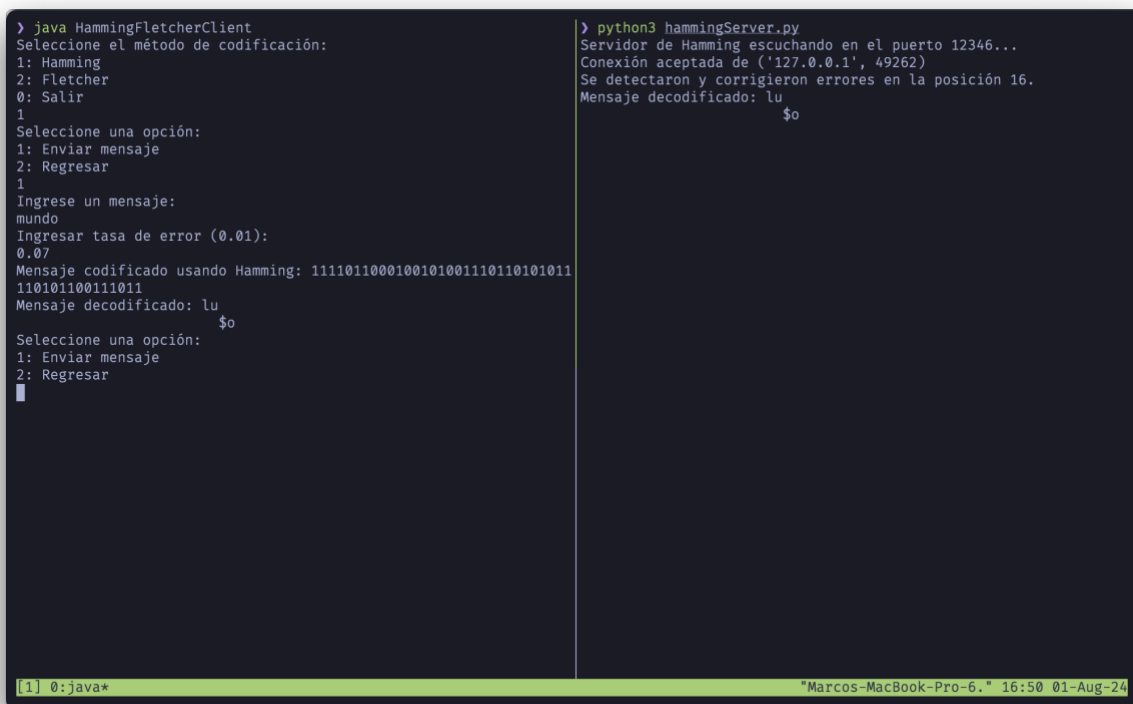
```
> java HammingFletcherClient
Seleccione el método de codificación:
1: Hamming
2: Fletcher
0: Salir
1
Seleccione una opción:
1: Enviar mensaje
2: Regresar
1
Ingrese un mensaje:
mundo
Ingrese tasa de error (0.01):
0.03
Mensaje codificado usando Hamming: 1111011000100101001110110101011
110101100111011
Mensaje decodificado: mundo
Seleccione una opción:
1: Enviar mensaje
2: Regresar
1
```

```
> python3 hammingServer.py
Servidor de Hamming escuchando en el puerto 12346...
Conexión aceptada de ('127.0.0.1', 49201)
Se detectaron y corrigieron errores en la posición 14.
Mensaje decodificado: mundo
```

[1] 0:java* *Marcos-MacBook-Pro-6.* 16:48 01-Aug-24

Prueba No.3

- Mensaje: mundo
- Cadena binaria: 1111011000100101001110110101011110101100111011
- Probabilidad de error: 0.07



```
> java HammingFletcherClient
Seleccione el método de codificación:
1: Hamming
2: Fletcher
0: Salir
1
Seleccione una opción:
1: Enviar mensaje
2: Regresar
1
Ingrese un mensaje:
mundo
Ingresar tasa de error (0.01):
0.07
Mensaje codificado usando Hamming: 1111011000100101001110110101011
110101100111011
Mensaje decodificado: lu
$0
Seleccione una opción:
1: Enviar mensaje
2: Regresar
1
```

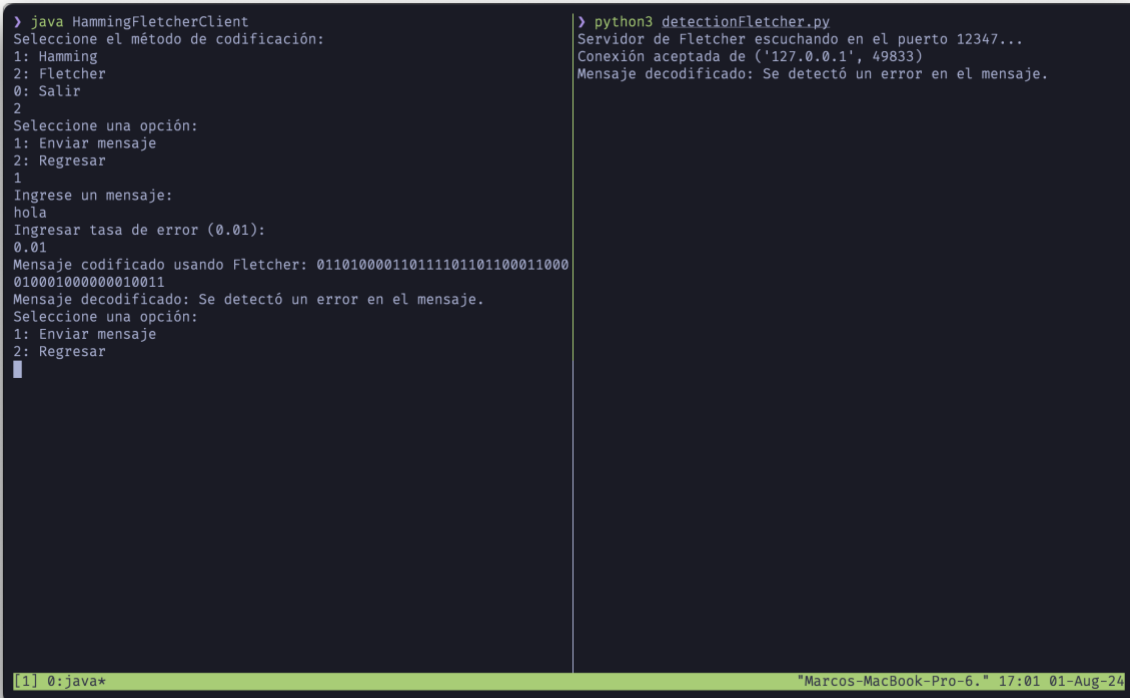
```
> python3 hammingServer.py
Servidor de Hamming escuchando en el puerto 12346...
Conexión aceptada de ('127.0.0.1', 49262)
Se detectaron y corrigieron errores en la posición 16.
Mensaje decodificado: lu
$0
```

[1] 0:java* *Marcos-MacBook-Pro-6.* 16:50 01-Aug-24

Fletcher checksum

Prueba No.1

- Mensaje: Hola
- Cadena binaria: 011010000110111101101100011000010001000000010011
- Probabilidad de error: 0.01



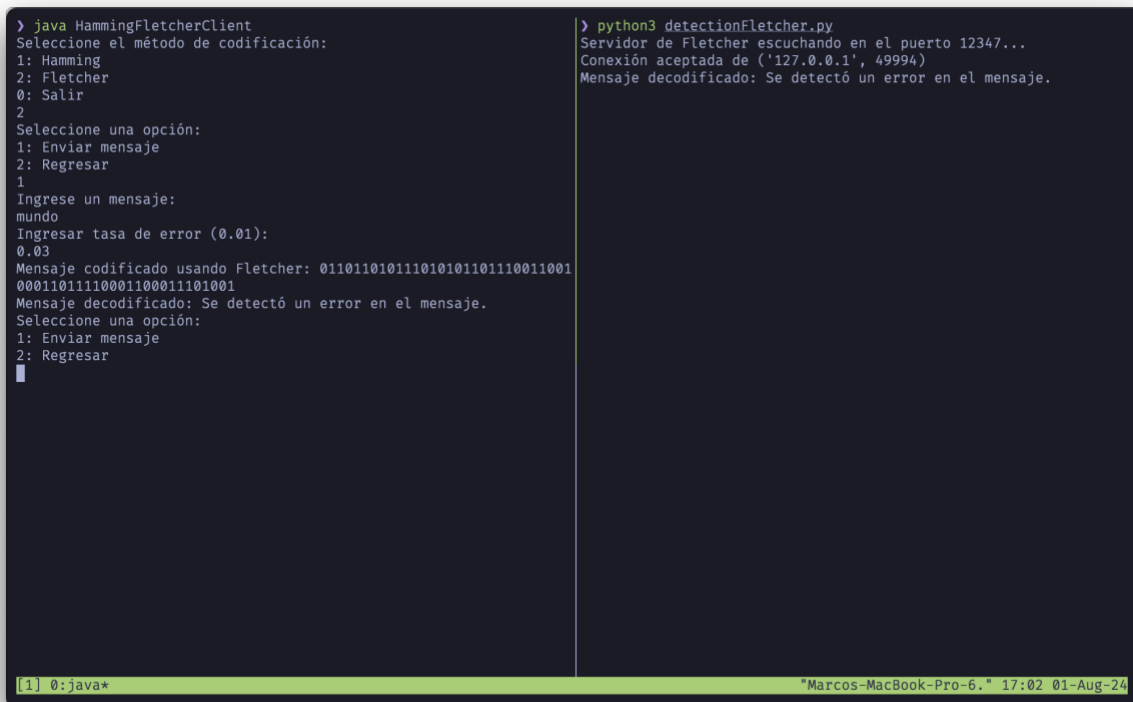
```
> java HammingFletcherClient
Seleccione el método de codificación:
1: Hamming
2: Fletcher
0: Salir
2
Seleccione una opción:
1: Enviar mensaje
2: Regresar
1
Ingrese un mensaje:
hola
Ingrese tasa de error (0.01):
0.01
Mensaje codificado usando Fletcher: 011010000110111101101100011000
010001000000010011
Mensaje decodificado: Se detectó un error en el mensaje.
Seleccione una opción:
1: Enviar mensaje
2: Regresar
█

> python3 detectionFletcher.py
Servidor de Fletcher escuchando en el puerto 12347...
Conexión aceptada de ('127.0.0.1', 49833)
Mensaje decodificado: Se detectó un error en el mensaje.
```

[1] 0:java* *Marcos-MacBook-Pro-6.* 17:01 01-Aug-24

Prueba No.2

- Mensaje: mundo
- Cadena binaria: 01101101011101010110111001100100011011110001100011101001
- Probabilidad de error: 0.03



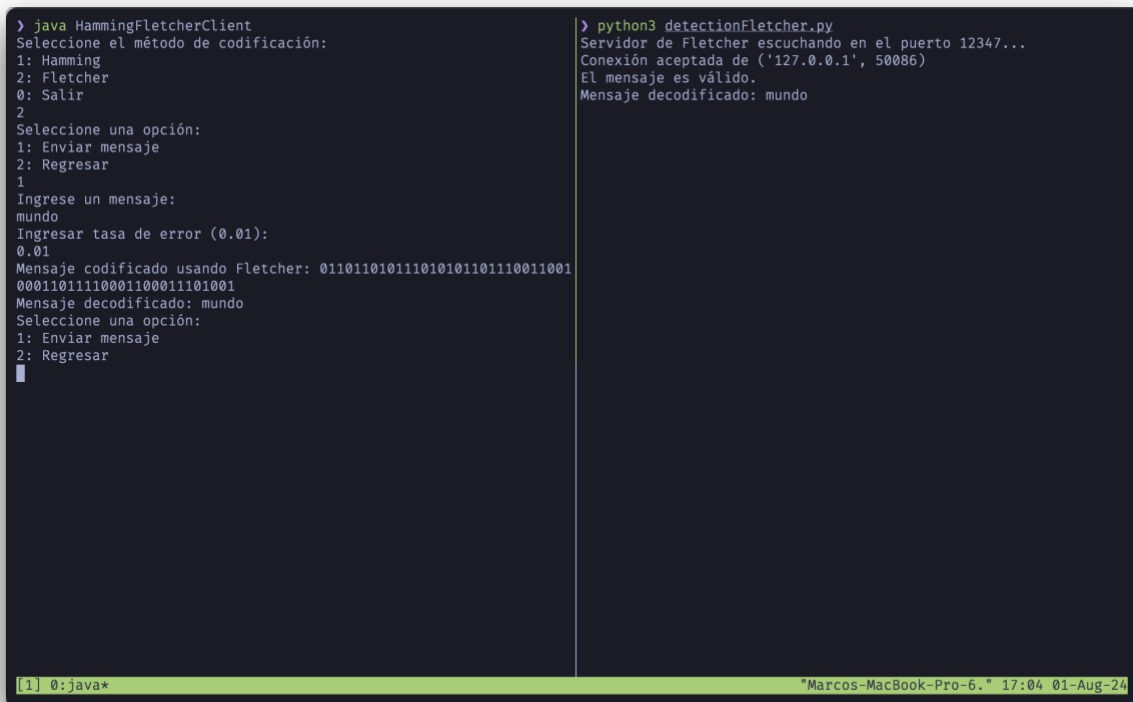
```
> java HammingFletcherClient
Seleccione el método de codificación:
1: Hamming
2: Fletcher
0: Salir
2
Seleccione una opción:
1: Enviar mensaje
2: Regresar
1
Ingrese un mensaje:
mundo
Ingresar tasa de error (0.01):
0.03
Mensaje codificado usando Fletcher: 011011010111010101101110011001
00011011110001100011101001
Mensaje decodificado: Se detectó un error en el mensaje.
Seleccione una opción:
1: Enviar mensaje
2: Regresar
█

> python3 detectionFletcher.py
Servidor de Fletcher escuchando en el puerto 12347...
Conexión aceptada de ('127.0.0.1', 49994)
Mensaje decodificado: Se detectó un error en el mensaje.
```

[1] 0:java* *Marcos-MacBook-Pro-6.* 17:02 01-Aug-24

Prueba No.3

- Mensaje: mundo
- Cadena binaria: 01101101011101010110111001100100011011110001100011101001
- Probabilidad de error: 0.01



```
> java HammingFletcherClient
Seleccione el método de codificación:
1: Hamming
2: Fletcher
0: Salir
2
Seleccione una opción:
1: Enviar mensaje
2: Regresar
1
Ingrese un mensaje:
mundo
Ingresar tasa de error (0.01):
0.01
Mensaje codificado usando Fletcher: 011011010111010101101110011001
00011011110001100011101001
Mensaje decodificado: mundo
Seleccione una opción:
1: Enviar mensaje
2: Regresar
1
```

```
> python3 detectionFletcher.py
Servidor de Fletcher escuchando en el puerto 12347...
Conexión aceptada de ('127.0.0.1', 50086)
El mensaje es válido.
Mensaje decodificado: mundo
```

[1] 0:java* *Marcos-MacBook-Pro-6.* 17:04 01-Aug-24

Discusión

En esta segunda parte del laboratorio, se diseñó un sistema de comunicación cliente-servidor con el objetivo de evaluar y comparar la eficacia de los algoritmos de corrección de errores Hamming y Fletcher. Este sistema tiene un cliente en Java y un servidor en Python, cada uno con funciones específicas para codificar, transmitir y decodificar mensajes, además de simular errores en la transmisión para probar los algoritmos.

El cliente, desarrollado en Java, tiene la responsabilidad de codificar los mensajes de texto en formato binario y aplicar uno de los dos algoritmos de corrección de errores seleccionados: Hamming o Fletcher. Además, el cliente introduce un nivel configurable de ruido en los mensajes, lo que permite simular errores típicos que pueden ocurrir durante la transmisión a través de la red. Este enfoque es crucial para probar cómo cada algoritmo maneja errores y garantiza la integridad del mensaje recibido.

El servidor en Python, por su parte, está configurado para recibir el mensaje codificado y con ruido desde el cliente. Dependiendo del algoritmo utilizado, el servidor decodifica y verifica el mensaje. Si se emplea el algoritmo de Hamming, el servidor corrige los errores detectados y reconstruye el mensaje original. Si se utiliza el algoritmo de Fletcher, el servidor verifica la integridad del mensaje y detecta cualquier discrepancia introducida por el ruido. Finalmente, el servidor envía de vuelta al cliente el mensaje decodificado o un mensaje de error si no se puede corregir el mensaje.

Durante las pruebas realizadas, se observó que el algoritmo de Hamming, aunque efectivo para corregir errores en condiciones de baja a moderada tasa de error, muestra limitaciones cuando se enfrenta a una mayor cantidad de errores. Esto se debe a que el código de Hamming

tiene una capacidad de corrección limitada, especialmente en entornos con alta tasa de error. Por último, el algoritmo de Fletcher, diseñado principalmente para la verificación de integridad, proporciona una buena detección de errores incluso en condiciones de ruido moderado, pero no corrige errores como lo hace Hamming. La diferencia en el rendimiento se debe a que Fletcher se basa en una suma de verificación que detecta errores pero no proporciona corrección, mientras que Hamming incluye mecanismos para corregir errores de forma activa.

Conclusiones

- El algoritmo de Hamming demostró ser efectivo en la corrección de errores durante la transmisión de mensajes con una tasa de error baja a moderada.
- El algoritmo de Fletcher, aunque no corrige errores, proporcionó una robusta capacidad de detección de errores.
- La introducción de ruido en los mensajes simuló de manera efectiva los errores que pueden ocurrir en una red real. Las pruebas demostraron cómo las tasas de error afectan la capacidad de los algoritmos para mantener la integridad del mensaje.

Bibliografía

GeekforGeeks. (11 de junio de 2020). *Hamming code Implementation in Java* . Obtenido de

Geek for Geeks: <https://www.geeksforgeeks.org/hamming-code-implementation-in-java/>

GeekforGeeks. (7 de mayo de 2024). *Error Detection Code checksum*. Obtenido de Geek for

geeks: <https://www.geeksforgeeks.org/error-detection-code-checksum/>

GeeksforGeeks. (17 de Junio de 2024). *Hamming Code in Computer Network*. Obtenido de Geek

for Geeks: <https://www.geeksforgeeks.org/hamming-code-in-computer-network/>

yadav, C. (30 de junio de 2020). *Fletcher's Checksum* . Obtenido de tutorial's point:

<https://www.tutorialspoint.com/fletcher-s-checksum>