UNIVERSIDAD DEL VALLE DE GUATEMALA

Redes, sección 20

LAB # 2

Diego Linares - 221256
Joaquin Campos – 22155

Guatemala, Julio 2025

Se implemtno un sistema de transmisión que permita detectar y corregir errores en datos enviados a través de una red, aplicando el algoritmo Hamming (11,7) utilizando sockets TCP para la comunicación entre un emisor (Java) y un receptor (Python).

Descripción del sistema desarrollado

Se implementó un flujo en 5 capas:

1. Capa de Aplicación:
    a. El usuario ingresa un mensaje en texto.
    b. Este mensaje se procesa para ser transmitido.
2. Capa de Presentación:
    a. El mensaje se convierte a su representación binaria ASCII (8 bits por carácter).
3. Capa de Enlace:
    a. Se aplica el algoritmo de codificación elegido:
        i. Hamming (11,7): agrega bits de paridad y permite corregir errores de 1 bit por bloque.
        ii. Fletcher Checksum: genera un checksum para detectar errores (no corrige).
    b. El usuario puede seleccionar el método antes de enviar.
4. Capa de Ruido:
    a. Se introduce ruido aleatorio (flip de bits) con una probabilidad configurable.
    b. Esto simula errores reales en transmisión.
5. Capa de Transmisión (Sockets TCP):
    a. Se utiliza un socket TCP para enviar la trama desde el emisor en Java al receptor en Python.
    b. El receptor escucha en el puerto definido y procesa los datos recibidos.

Algoritmos implementados

Hamming (11,7)

- Implementado en Java para el emisor.
- Corrige errores de 1 bit por bloque de 11 bits.

- El receptor Python recibe los bloques, detecta errores mediante el síndrome, los corrige y reconstruye el mensaje original.

Checksum Fletcehr

- Implementado en python para el emisor.
- Hacemos el pading, y acumulamos los dos valores, luego los combinamos en bloques para generar el checksum
- El receptor Java si al recibir los datos vuelve a calcular el mismo checksum y concuerda, se asume que los datos no tienen errores.

Uso de sockets

- Tipo de socket: TCP (stream)
- Lado emisor (Java): crea un socket cliente y envía los datos binarios.
- Lado receptor (Python): crea un socket servidor que escucha en un puerto (ej. 6543) y procesa la trama recibida.

¿Por qué TCP?

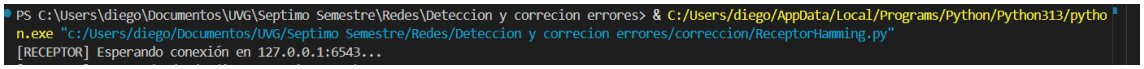
- TCP garantiza la entrega ordenada de datos sin pérdidas, lo que facilita el enfoque del laboratorio donde se evalúan errores intencionales introducidos solo por el ruido, no por la red.

Flujo de ejecución

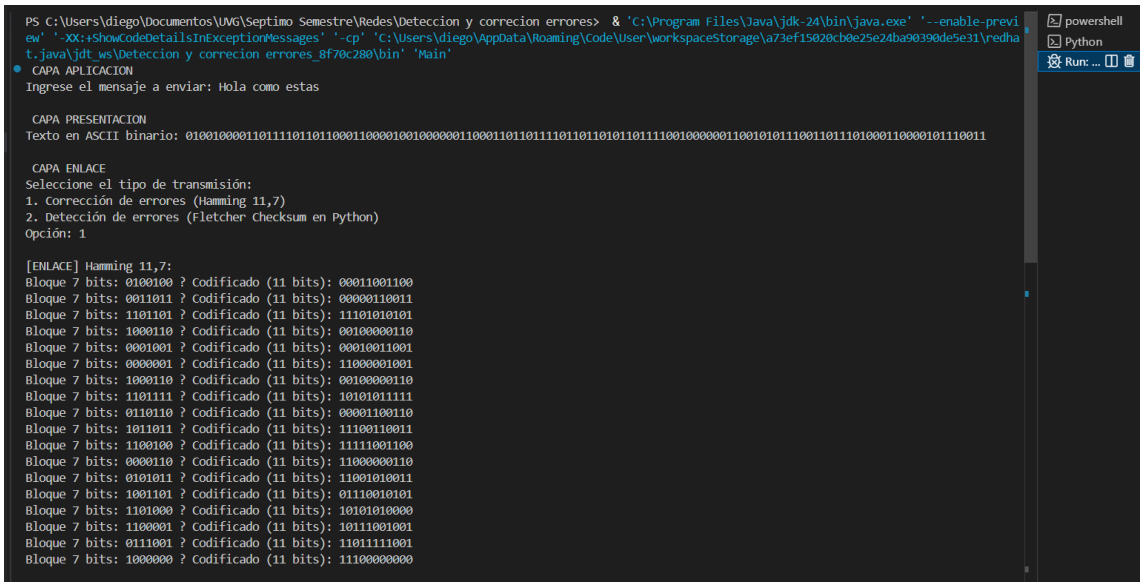
1. El usuario ejecuta primero el receptor Python (escucha).
2. Se corre el emisor Java, que:
    a. Convierte texto a binario
    b. Codifica con Hamming (o llama Fletcher)
    c. Aplica ruido
    d. Envía la trama vía socket TCP
3. El receptor recibe la trama, la procesa y:
    a. Si es Hamming, corrige los errores detectados y reconstruye el mensaje ASCII original.
    b. Si es Fletcher, detecta errores y los reporta.

Evidencias:

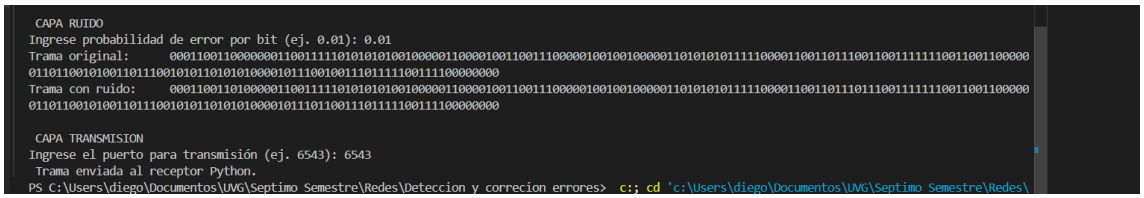
Se ejecuta el receptor y queda a la espera del mensaje:



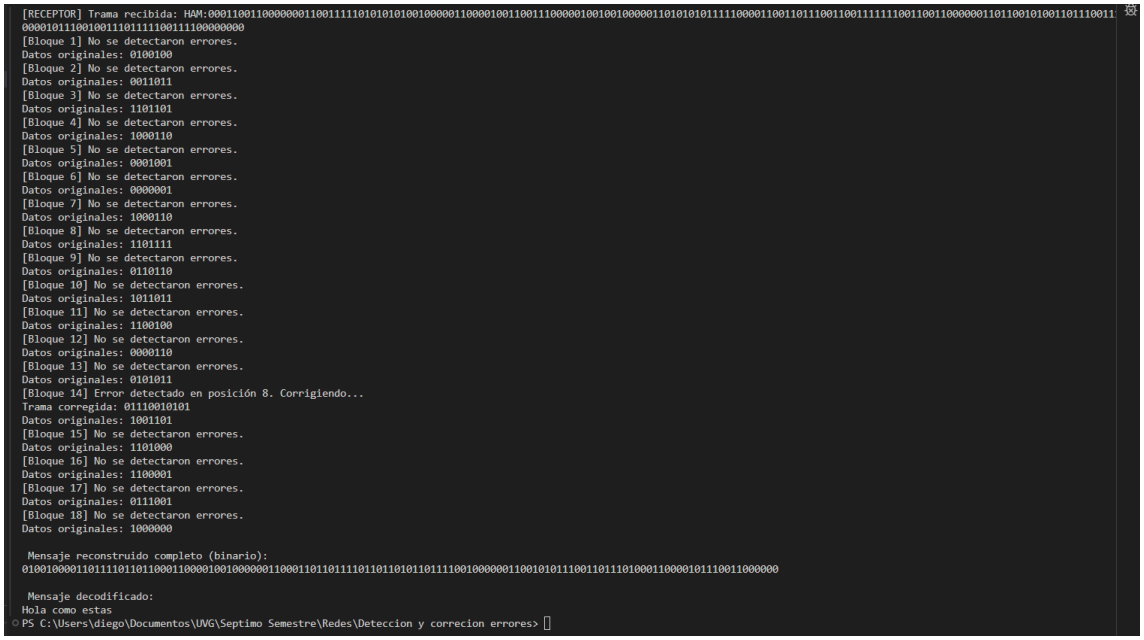
Luego el emisor se corre y envia el mensaje:



Va pasando por las 5 capas..



Finalmente es enviado y ahora el receptor recibe:



Pruebas y Resultados

Nº	Algoritmo	Bloque (Fletcher)	Largo mensaje (char)	Prob. error	Bits datos	Bits enviados	Overhead (%)	Esperado	Obtenido
1	Hamming	—	2	0	16	24	50%	recibe	recibe
2	Hamming	—	2	0.05	16	24	50%	error	recibe
3	Hamming	—	8	0	64	96	50%	recibe	recibe
4	Hamming	—	8	0.1	64	96	50%	error	ERROR
5	Hamming	—	16	0.01	128	192	50%	recibe	recibe
6	Hamming	—	16	0.05	128	192	50%	error	ERROR
7	Hamming	—	2	0	16	32	100%	recibe	recibe
8	Hamming	—	2	0.1	16	32	100%	error	ERROR
9	Hamming	—	8	0	64	80	25%	recibe	recibe
10	Hamming	—	8	0.05	64	80	25%	error	ERROR
11	Fletcher	8	16 (determinacionfinal)	0.01	128	144	12.50%	recibe	error
12	Fletcher	8	16 (comunicacionsegura)	0.1	128	144	12.50%	error	error
13	Fletcher	16	4 (hola)	0	32	64	100%	recibe	recibe
14	Fletcher	16	4 (test)	0.05	32	64	100%	error	error
15	Fletcher	16	12 (ingenieriaUVG)	0	96	112	16.70%	recibe	recibe
16	Fletcher	16	12 (algoritmodatos)	0.1	96	112	16.70%	error	error

17	Fletcher	32	8 (proteina)	0	64	96	50%	recibe	error
18	Fletcher	32	8 (checksum)	0.05	64	96	50%	error	error
19	Fletcher	32	16 (universidad uvg)	0.01	128	160	25%	recibe	error
20	Fletcher	32	16 (transmisio ndatos)	0.1	128	160	25%	error	error

El cáclulo del Overhead fue de la siguiente forma:

$$((\text{Bits enviados} - \text{Bits datos}) / \text{Bits datos}) \times 100$$

Imagenes de pruebas

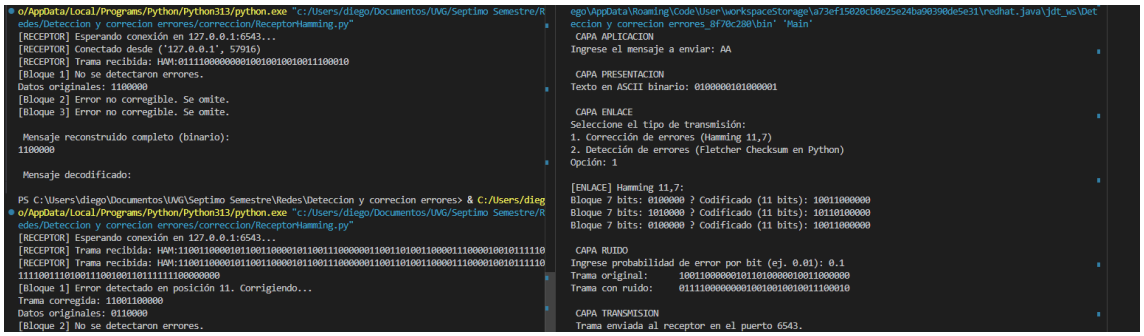
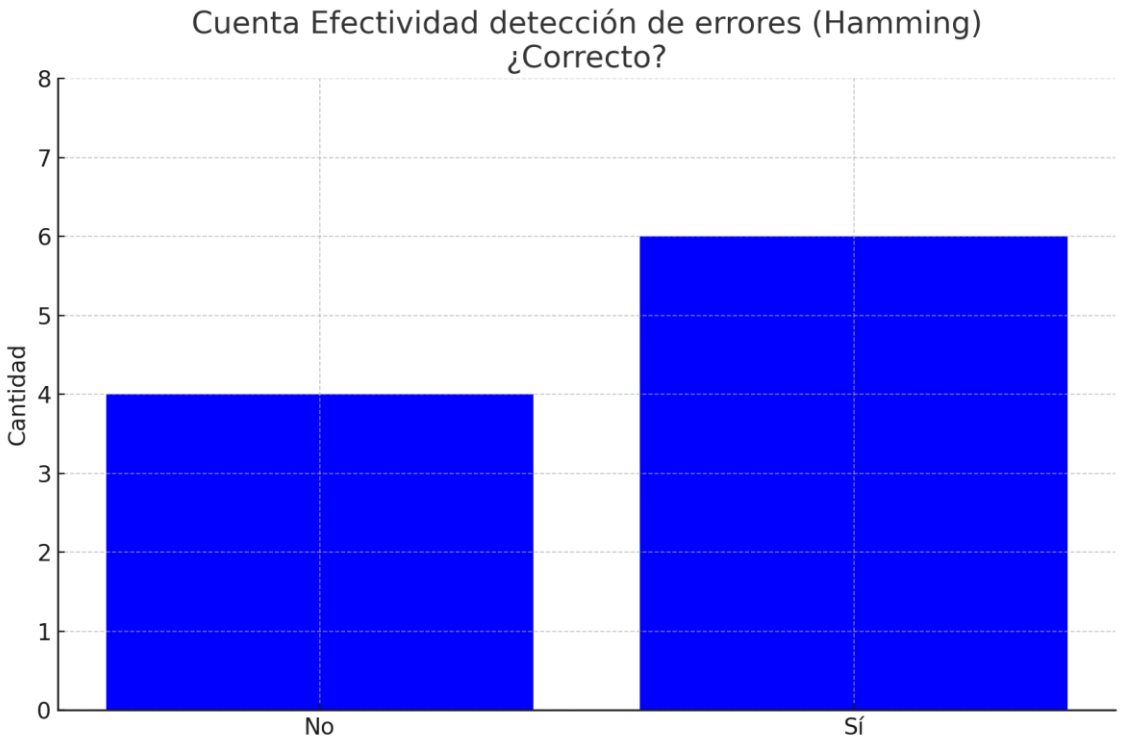
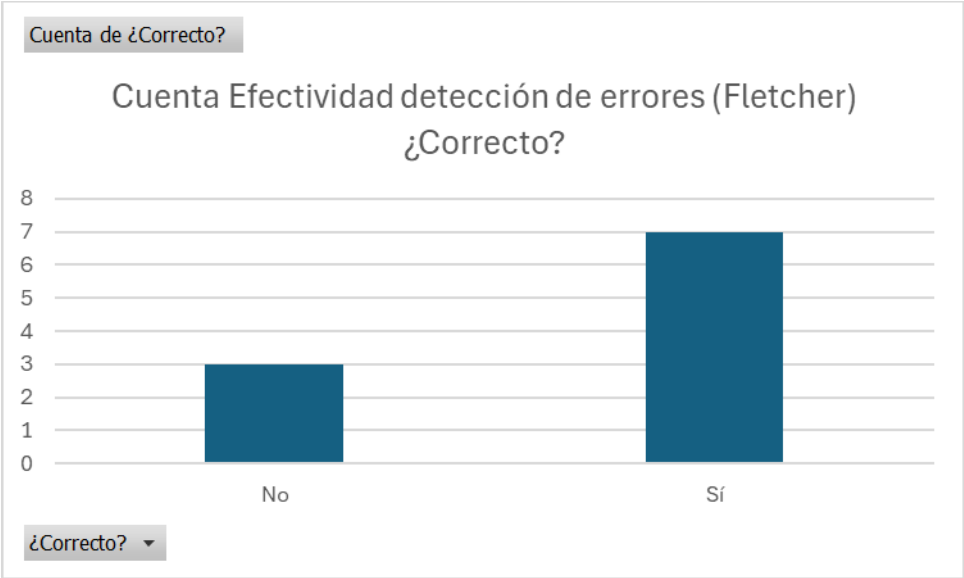


Imagen de arriba muestra el error al ponerle 0.1 de ruido en un mensaje de longitud 2

Grafica Hamming:



Grafica Fletcher:



Discusión

En las pruebas realizadas con el algoritmo Fletcher, observamos que logra detectar correctamente la mayoría de los errores cuando estos ocurren en la transmisión de datos, incluso con tasas de error bajas como 0.01 (1%). Sin embargo, también se

encontraron algunos casos donde el algoritmo no logró detectar el error, lo cual indica que no es infalible o que el ruido no siempre afecta.

Cuando las tasas de error son bajas o la retransmisión es posible y no costosa, es mejor usar un algoritmo de detección de errores como Fletcher. En cambio, cuando no es posible retransmitir, o si la confiabilidad debe ser muy alta (como en transmisiones espaciales o en tiempo real), entonces conviene usar algoritmos de corrección de errores como Hamming, que pueden recuperar los datos sin necesidad de repetir la transmisión.

En las pruebas realizadas con el algoritmo de corrección de errores Hamming (11,7), se observó que el algoritmo fue efectivo en la mayoría de los casos cuando la probabilidad de error fue baja o cuando solo se introdujo un único bit erróneo por bloque. Esto se evidenció en las pruebas donde, a pesar de existir ruido, el receptor logró corregir el error y reconstruir el mensaje original.

Sin embargo, en los escenarios con probabilidad de error mayor (0.05 y 0.1) y mensajes largos, aparecieron varios casos donde el algoritmo no pudo corregir los errores. Esto se debe a que Hamming (11,7) solo puede corregir un error por bloque de 11 bits y no detecta ni corrige errores múltiples en un mismo bloque. Así, en mensajes largos o cuando el ruido altera más de un bit por bloque, el algoritmo falla.

Esto explica por qué, aunque el algoritmo tiene una alta tasa de aciertos (6 de 10 casos correctos en la tabla), no es infalible en canales con ruido significativo.

Conclusiones

El algoritmo Fletcher Checksum detectó correctamente la mayoría de los errores introducidos, mostrando una buena efectividad en contextos con errores leves o moderados.

La detección de errores sin corrección es útil cuando se puede volver a enviar el mensaje, permitiendo sistemas más ligeros y rápidos, pero menos robustos ante errores múltiples.

El algoritmo Hamming (11,7) corrige eficazmente errores de un solo bit en los bloques transmitidos, garantizando la recuperación de datos cuando la tasa de error es baja. Su efectividad disminuye cuando se incrementa la probabilidad de

error o cuando el ruido afecta múltiples bits dentro del mismo bloque, ya que no fue diseñado para manejar errores múltiples.

Referencia

Fletcher, J. G. (1982). An arithmetic checksum for serial transmission. IEEE Transactions on Communications, https://doi.org/10.1109/TCOM.1982.1095460

OpenWebinars. (2019, marzo 28). *Codificación Hamming (Detección y corrección de errores)* [Video]. YouTube. https://www.youtube.com/watch?v=gQK9nROFX20

Jarroba. (2014, marzo 9). *Código de Hamming: Detección y corrección de errores*. Jarroba. https://jarroba.com/codigo-de-hamming-deteccion-y-correccion-de-errores/