UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Laboratorio 2 – Parte 1

Esquemas de detección y corrección de errores

Presentado por:

Sofía Lam Méndez – 21548
Eber Alexander Cuxé Tan – 22648

Redes

Guatemala, julio de 2025

1. **Link al repositorio: https://github.com/SofiLam13/RedesLab2.git**

2. Resumen

Este laboratorio tuvo como objetivo principal la implementación y comparación de dos algoritmos de detección y corrección de errores, Hamming y Fletcher, a través de una arquitectura cliente-servidor que utilizaba sockets TCP. El cliente, desarrollado en Python, fue el encargado de generar y codificar mensajes binarios aleatorios con ambos algoritmos, introduciendo errores de manera controlada para simular diferentes escenarios. Por su parte, el servidor, programado en C, se dedicó a recibir estas tramas, verificar su integridad y responder según la validez del mensaje.

Para evaluar el rendimiento y la fiabilidad de los métodos, se realizaron más de 1000 pruebas exitosas por cada algoritmo, variando las probabilidades de error. Los resultados obtenidos se registraron en archivos CSV para su posterior análisis estadístico en MATLAB. Las métricas clave consideradas para esta evaluación incluyeron el tiempo de transmisión, la frecuencia de detección de errores y el porcentaje de mensajes válidos, proporcionando una visión integral del desempeño de cada algoritmo.

3. Descripción y metodología utilizada:

Este laboratorio tuvo como objetivo implementar y evaluar esquemas de detección y corrección de errores en una arquitectura de capas que simula la transmisión de información sobre un canal no confiable. Se trabajó con los algoritmos de Hamming (corrección) y Fletcher-16 (detección), tanto emisor como receptor a través de Python y C respectivamente.

A partir de esto, se realizaron pruebas de forma automática, con mensajes aleatorios generados automáticamente los cuales sometidos a distintos niveles de ruido. Dichas pruebas se ejecutaron en grandes cantidades, lo cual permitió analizar los siguiente:

- La tasa de errores detectados o corregidos.
- El porcentaje de éxito por algoritmo.
- El tiempo promedio de ejecución.
- La distribución de errores según la probabilidad.

Arquitectura y funcionamiento del sistema

El sistema se construyó con una arquitectura cliente-servidor utilizando sockets TCP. El emisor, programado en Python, enviaba tramas estructuradas al servidor receptor, desarrollado en C. El formato de las tramas era *algoritmo|block_size|trama_binaria*.

Para la codificación, el algoritmo Hamming operaba sobre bloques de 4 bits de datos, generando 7 bits con paridad. Fletcher, por su parte, trabajaba con bloques de 8 y 16 bits,

añadiendo un checksum de dos bloques al final de cada trama. Para simular condiciones de ruido en la transmisión, se inyectaron errores aleatorios en cada trama con una probabilidad entre 0.0 y 0.2.

Recolección y análisis de datos

El emisor se encargó de registrar métricas clave como el tiempo de envío y recepción, la presencia y posición del error inyectado, y la validez del mensaje según la respuesta del receptor. Todos los datos se almacenaron automáticamente en archivos CSV. Posteriormente, se realizó un análisis estadístico en MATLAB. Este análisis incluyó la generación de histogramas de probabilidad de error, gráficos de tiempo promedio y diagramas de pastel de validez, permitiendo comparar el rendimiento de ambos algoritmos, sin diferenciar por el tamaño de bloque en el caso de Fletcher.

4. Resultados

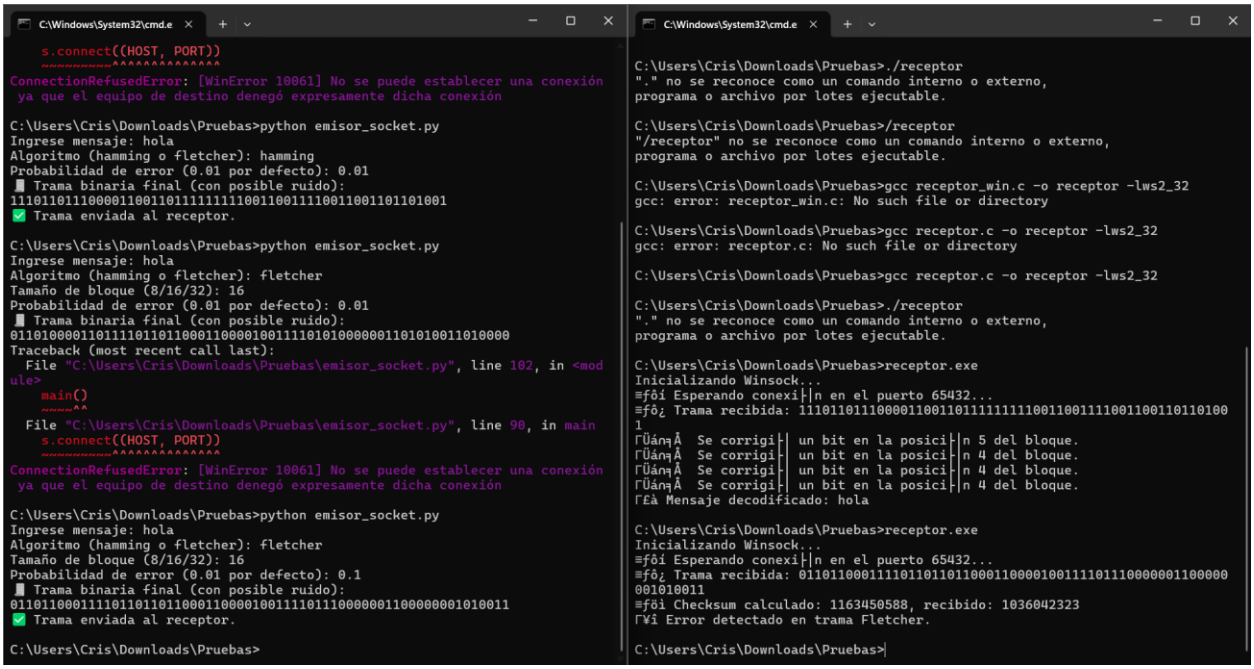

Figura 1. Códigos Unificados, Modo Manual.

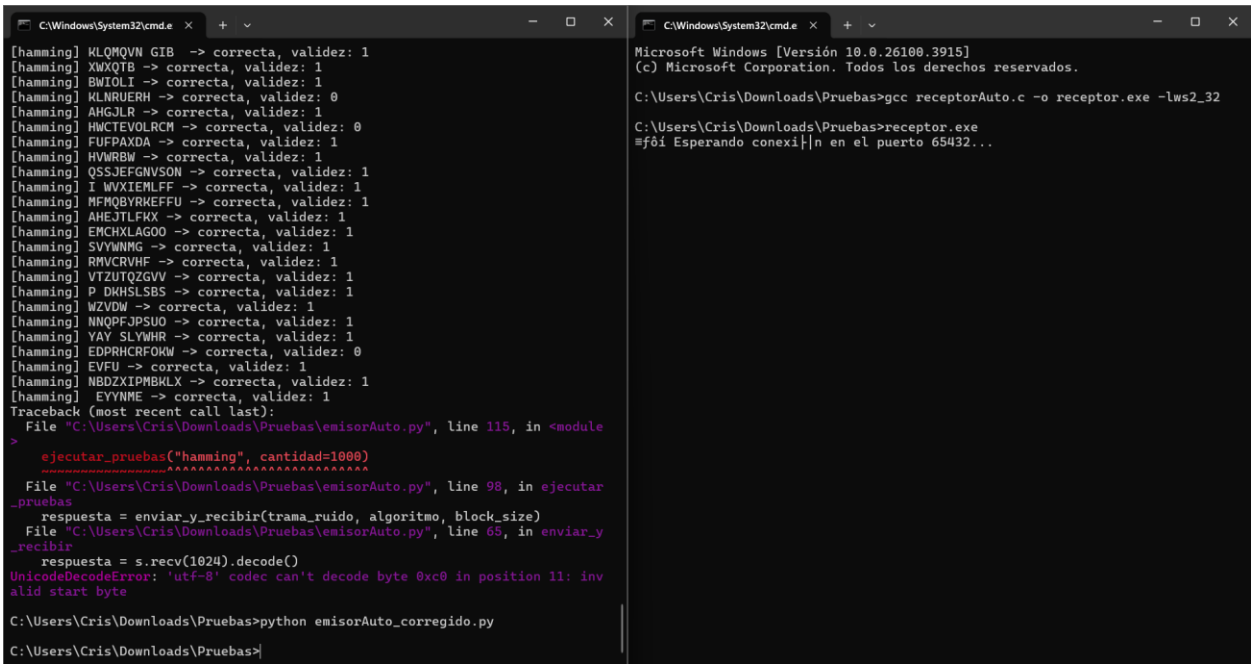

Figura 2. Conexión entre códigos y creación de CSV.

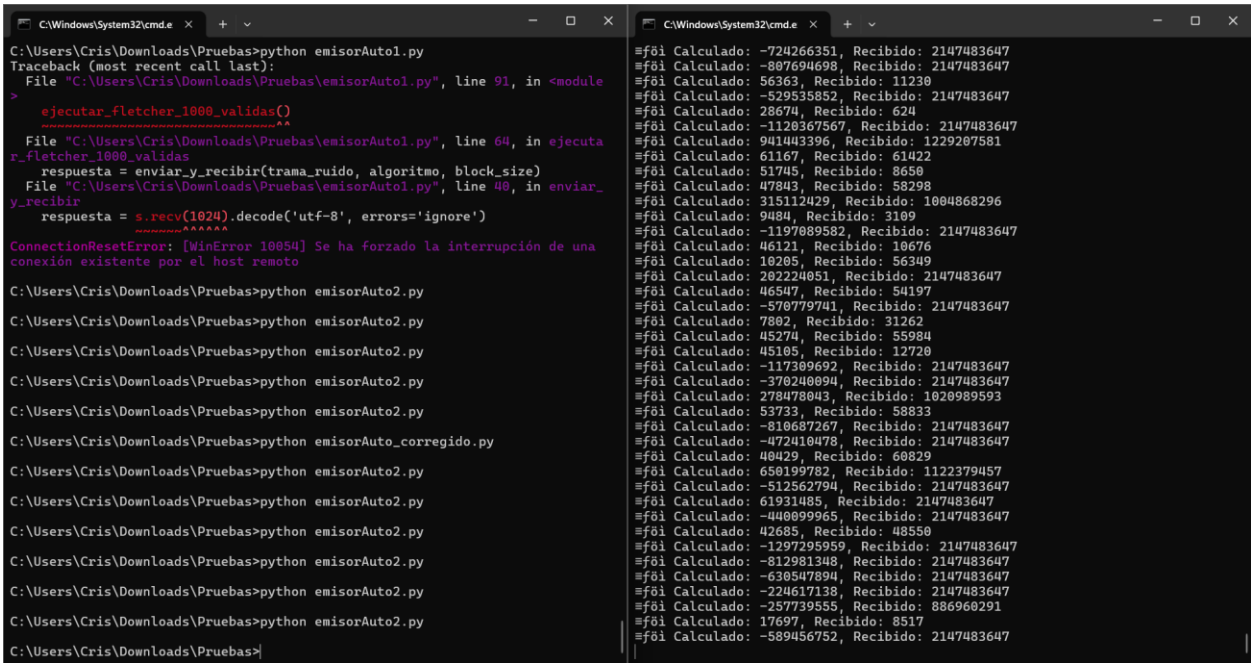

Figura 3. Corriendo pruebas para Fletcher-Checksum.

a. **Distribución de errores por algoritmo:**

i. **Hamming:**

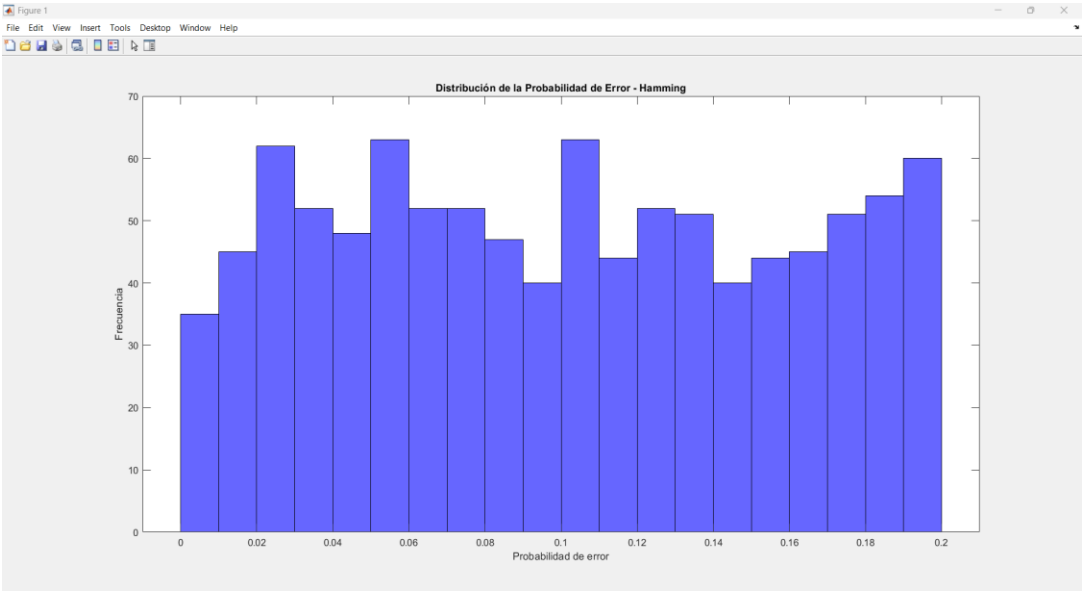

Figura 4. Recopilación probabilidades de error usadas, Hamming

ii. **Fletcher:**

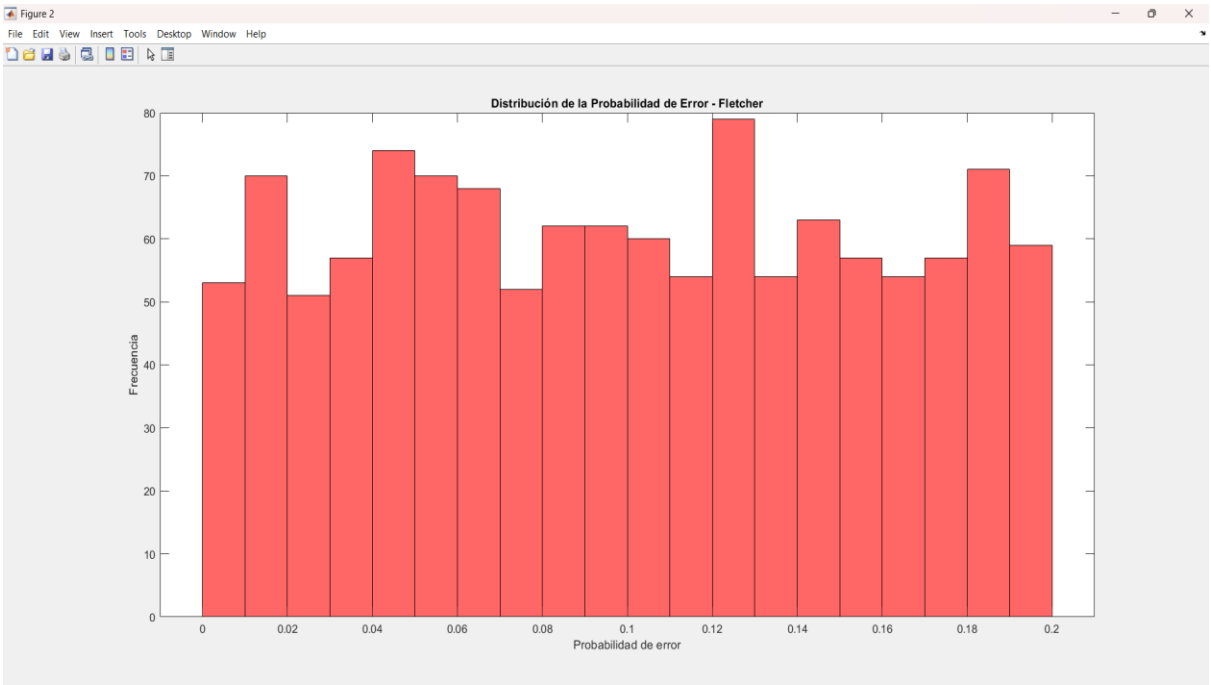

Figura 5. Recopilación probabilidades de error usadas, Fletcher.

b. **Validación de mensajes**

i. Hamming:

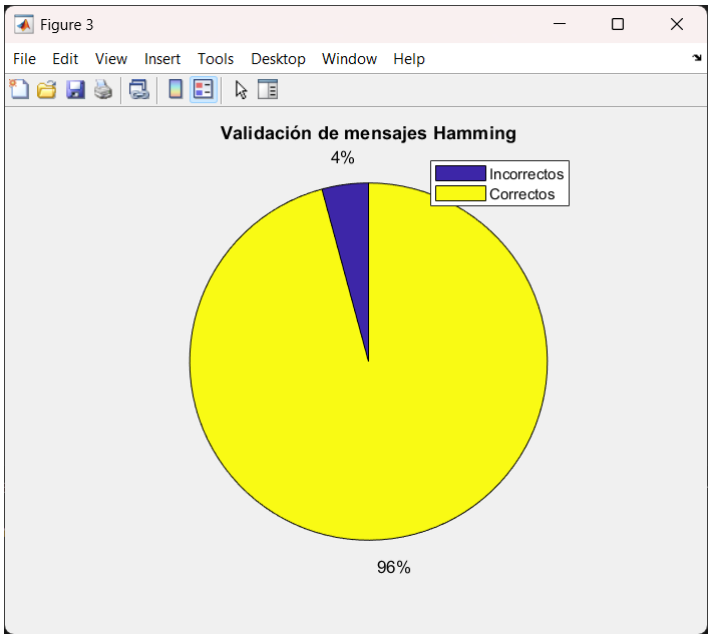

Figura 6. Validación de mensajes, Hamming.

ii. Fletcher:

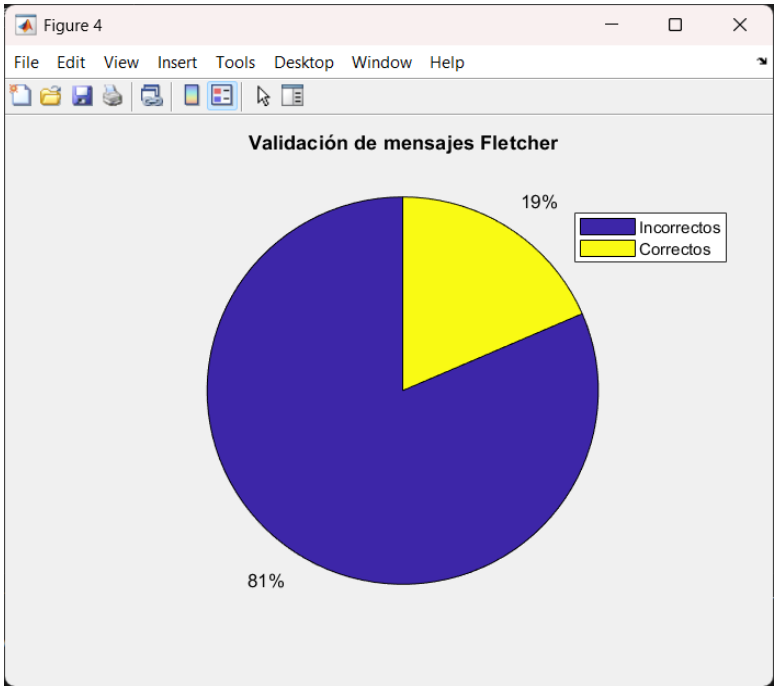

Figura 7. Validación de mensajes, Fletcher.

c. Rendimiento:

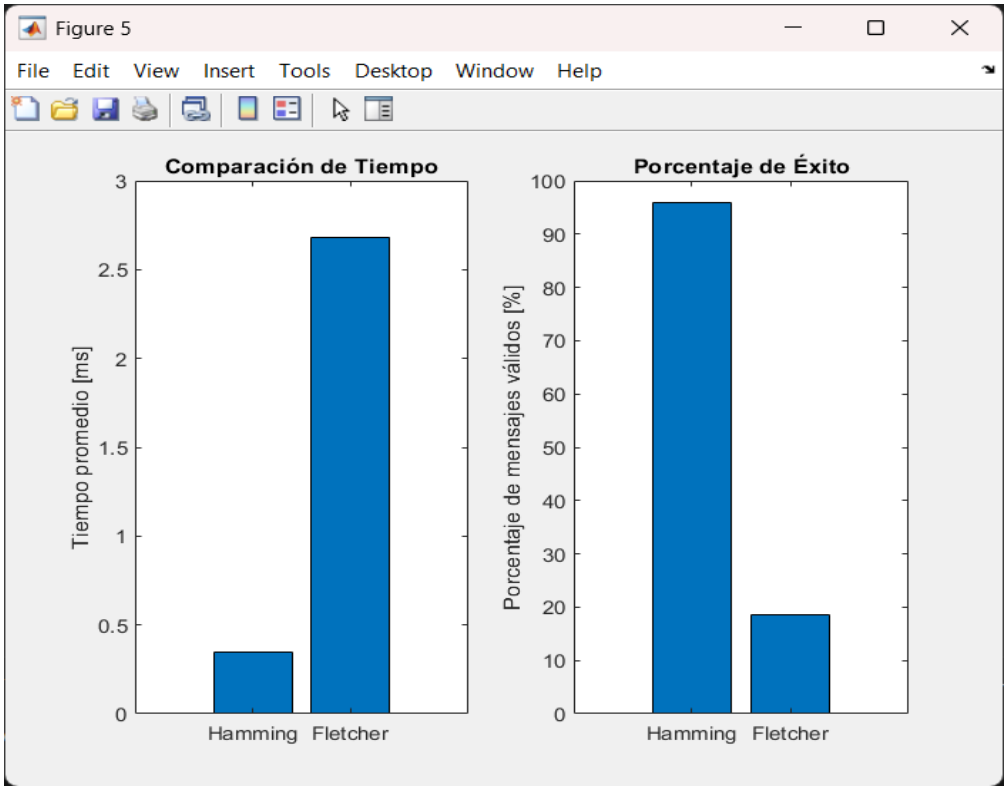

Figura 8. Comparación de rendimiento general entre algoritmos.

5. **Discusión:**

   De los resultados anteriores, se observa que, en eficiencia y tiempo, el algoritmo de Hamming mostró un desempeño significativamente más rápido que Fletcher, con un tiempo promedio de ejecución de 0.346 ms por mensaje, en comparación con los 2.683 ms de Fletcher. Esta diferencia se explica porque Hamming realiza operaciones más simples y directas sobre bits de paridad, mientras que Fletcher requiere cálculos aritméticos acumulativos más complejos.

   Respecto a la precisión, Hamming logró corregir exitosamente un alto porcentaje de errores, alcanzando una tasa de éxito del 96 %. Por su parte, Fletcher, al ser únicamente un algoritmo de detección y no de corrección, tuvo una efectividad menor: solo el 19 % de los mensajes fueron correctamente validados bajo condiciones de ruido moderado.

   En cuanto a la robustez, Hamming demostró mayor resistencia ante errores de un solo bit, siendo capaz de detectarlos y corregirlos adecuadamente. Fletcher, en cambio, presentó una proporción considerable de errores no detectados o mal validados cuando se introducían múltiples bits corruptos, lo que limita su confiabilidad.

6. **Comentario grupal:**

Este laboratorio nos ayudó a ver la importancia de seleccionar el algoritmo correcto de acuerdo con las necesidades del sistema. Si se requiere detección rápida y liviana, Fletcher puede ser útil. Sin embargo, si es crítica la corrección automática de errores, Hamming resulta ampliamente superior.

Este laboratorio también fomentó habilidades en el manejo de sockets, ya que nunca los habíamos utilizado; manipulación de bits; codificación binaria y herramientas de análisis estadístico como MATLAB; nos permitió aplicar conocimientos teóricos, y también desarrollar un flujo de trabajo automatizado.

7. Conclusiones:

a. Hamming es más eficiente en canales con errores de 1 bit, por lo que es recomendable en la mayoría de los escenarios prácticos donde el ruido es bajo o moderado.
b. Fletcher es menos robusto ante múltiples errores y no puede corregirlos, por lo que su aplicación debe limitarse a entornos controlados o como complemento a otras técnicas.
c. La correcta implementación de los algoritmos de comunicación en distintas plataformas (Python y C) demuestra que es posible integrar diferentes lenguajes en un entorno de red local con protocolos simples y efectivos.
d. La automatización de pruebas permitió evaluar objetivamente el rendimiento de ambos algoritmos en condiciones controladas.