Universidad del Valle de Guatemala
Redes CC3067
Kenneth Galvez 20079
Natanael Giron 20029

Laboratorio 2 - Segunda parte
Esquemas de detección y corrección de errores

Descripción de la práctica y metodología utilizada

En un entorno de comunicación real, el ruido y los errores son inevitables, lo que hace que la detección y corrección de errores sean esenciales para garantizar la integridad de los datos transmitidos. La segunda parte de este laboratorio se centra en la implementación y evaluación de un sistema de transmisión y recepción de mensajes que incorpora esquemas de detección y corrección de errores.

El objetivo principal en esta parte fue construir una aplicación que simula la transmisión y recepción de mensajes en un canal de comunicación susceptible a errores y ruido. La aplicación se basa en un modelo de capas con diferentes servicios, lo que permite una estructura modular para gestionar la codificación, transmisión, detección y corrección de errores, y finalmente, la presentación y aplicación de los mensajes.

Continuamos utilizando Código de Hamming y El codigo de Fletcher Checksum agregando a nuestra ecuación los Sockets. Sockets son una interfaz de programación que se utiliza para la comunicación entre procesos en diferentes capas de red. Los sockets se utilizan comúnmente para la interacción entre cliente y servidor. Los sockets proporcionan una serie de primitivas que están orientadas a ser utilizadas por los usuarios para establecer una conexión. (El Interfaz Socket, 2023)

La metodología utilizada en esta práctica fueron los siguientes pasos:

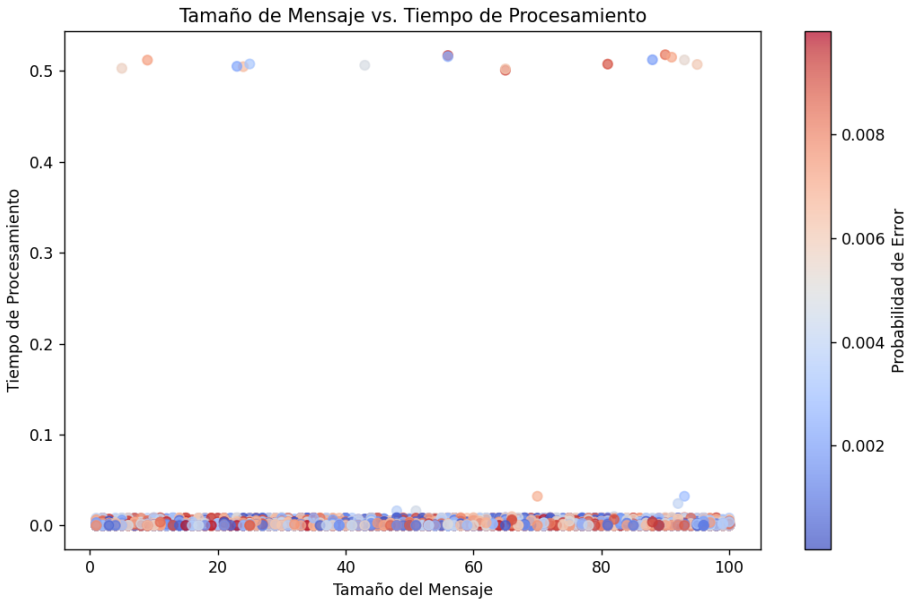
- Implementación de Algoritmos de Detección y Corrección (Realizados en la primera parte)
- Desarrollo de la Aplicación de Transmisión y Recepción
- Realización de Pruebas
- Análisis Estadístico y Generación de Gráficas

Universidad del Valle de Guatemala
Redes CC3067
Kenneth Galvez 20079
Natanael Giron 20029

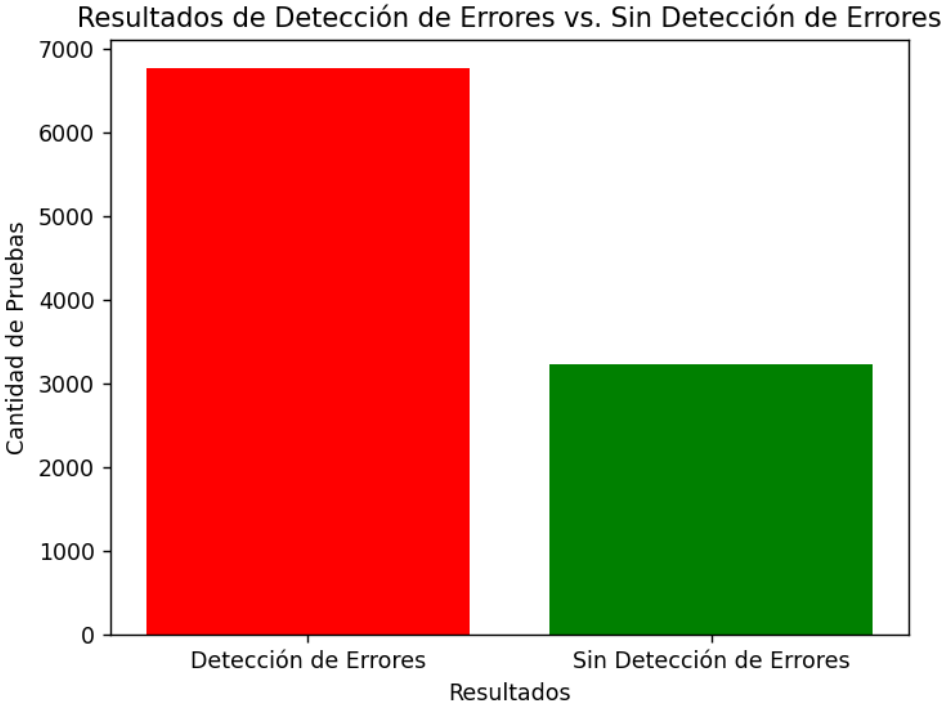
Resultados

Fletcher Checksum (Emisor: Python | Receptor: JavaScript)

Gráfica 1. Tamaño de Mensaje vs. Tiempo de Procesamiento (10,000 pruebas)



Gráfica 2. Detección de errores vs Sin Detección de errores (10,000 pruebas)

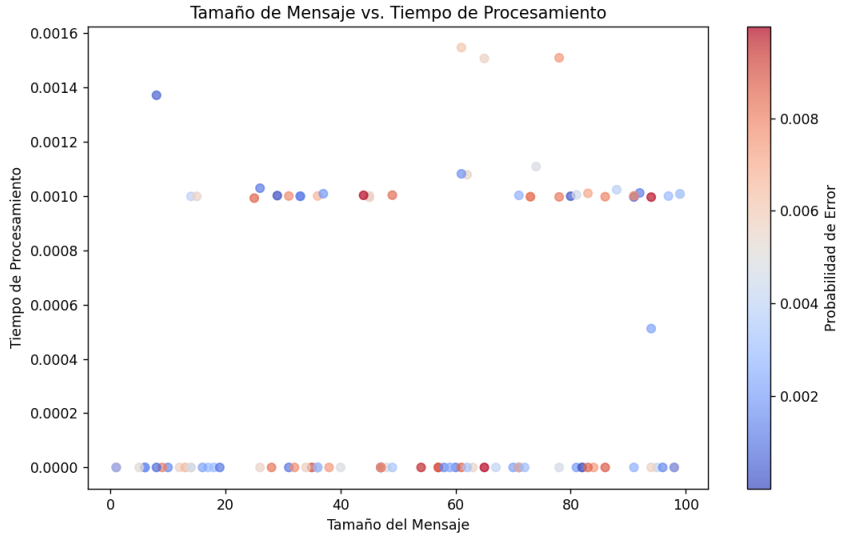


Universidad del Valle de Guatemala
Redes CC3067
Kenneth Galvez 20079
Natanael Giron 20029

Imagen 1. Tasa de detección de errores (10,000 pruebas)

```
PS D:\Github\Lab2.1Redes\Lab2.2 Fletcher> & C:/Users/kenic/AppData/Local/Programs/Python/Python311/python.exe "d:/Github/Lab2.1Redes/Lab2.2 Fletcher/emisor.py"
Tasa de detección de errores: 0.6767
```

Gráfica 3. Tamaño de Mensaje vs. Tiempo de Procesamiento (100 pruebas)



Gráfica 4. Detección de errores vs Sin Detección de errores (100,000 pruebas)

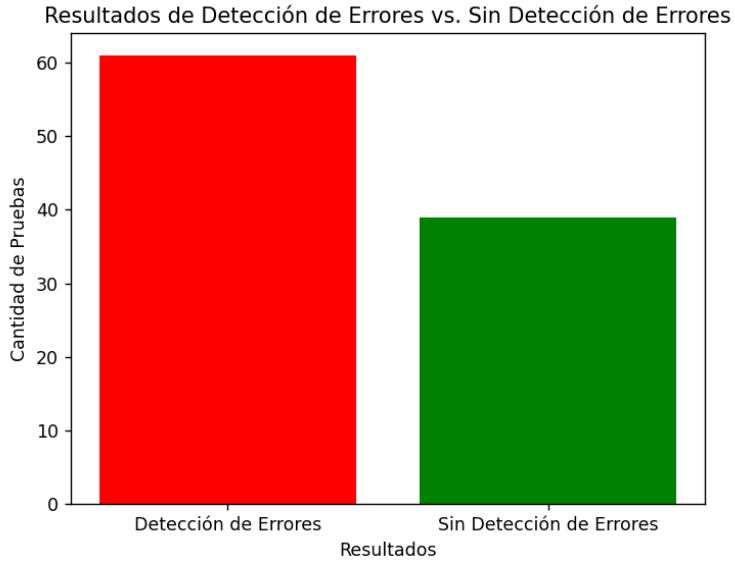


Imagen 2. Tasa de detección de errores (100,000 pruebas)

```
PS D:\Github\Lab2.1Redes\Lab2.2 Fletcher> & C:/Users/kenic/AppData/Local/Pr
ograms/Python/Python311/python.exe "d:/Github/Lab2.1Redes/Lab2.2 Fletcher/e
misor.py"
Tasa de detección de errores: 0.61
```

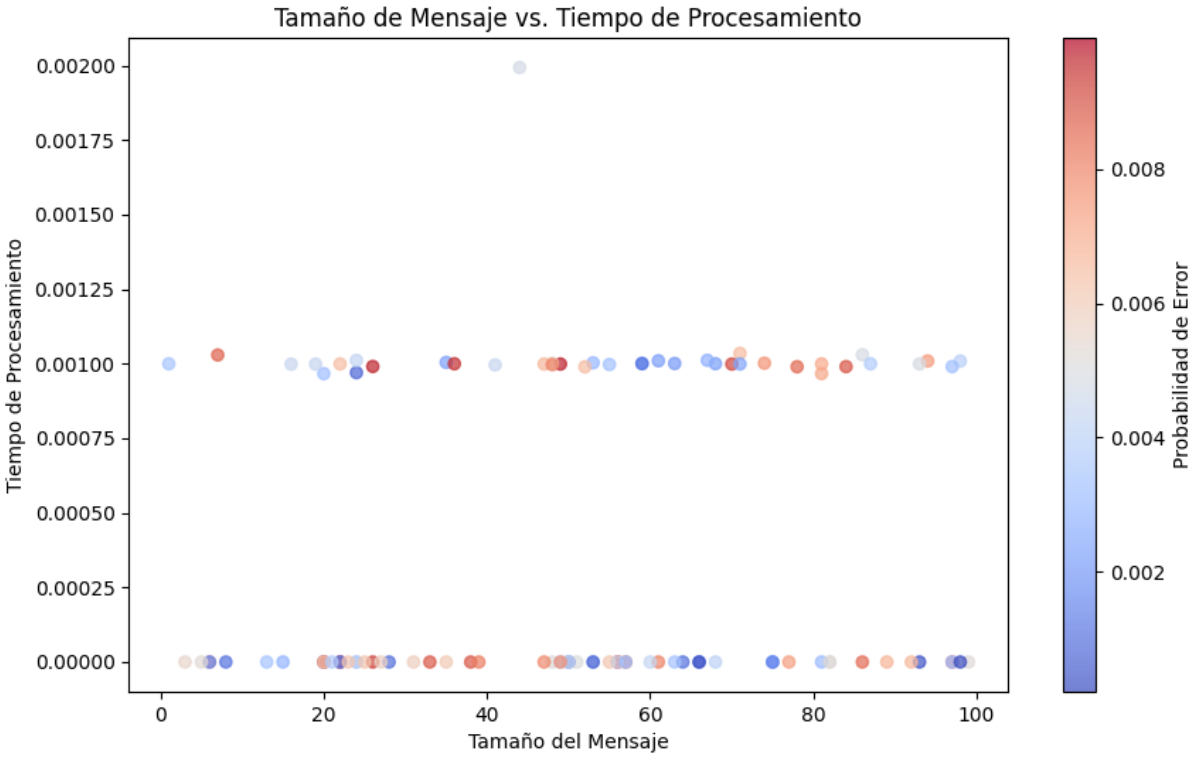
Código de Hamming(Emisor: Python | Receptor: JavaScript)

Tasa de Detección de errores para (10,000 pruebas)

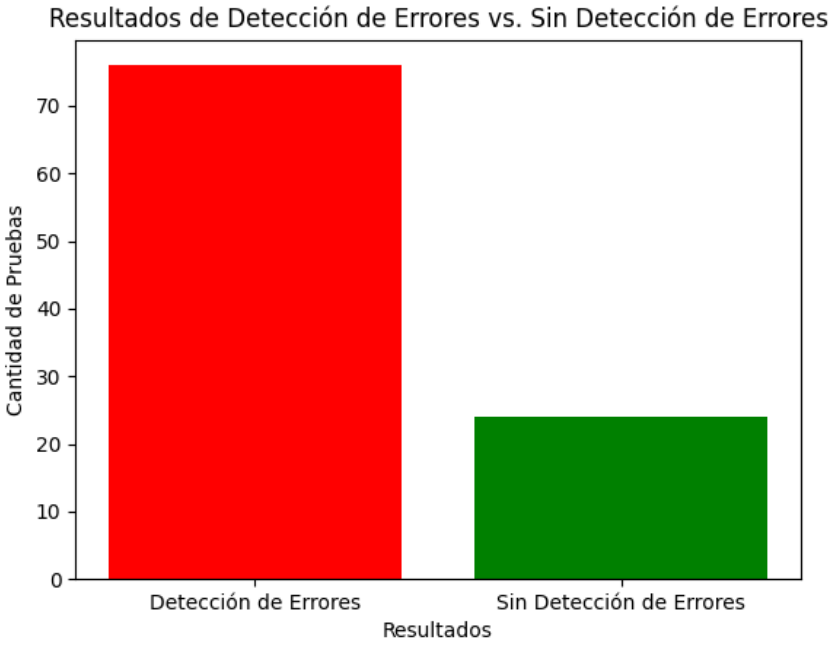
```
PS D:\Documentos\UVG\4 Cuarto Año\segundo semest
re\Redes\Lab2.1Redes> & C:/Users/rolan/AppData/L
ocal/Programs/Python/Python39/python.exe "d:/Doc
umentos/UVG/4 Cuarto Año/segundo semestre/Redes/
Lab2.1Redes/Lab2.2 Hamming/emisor2.py"
Tasa de detección de errores: 0.11
```

Universidad del Valle de Guatemala
Redes CC3067
Kenneth Galvez 20079
Natanael Giron 20029

Gráfica 5. Tamaño de Mensaje vs. Tiempo de Procesamiento (10,000 pruebas)



Gráfica 6. Resultados de Detección de Errores o Sin Detección de errores (10,000 pruebas)



Universidad del Valle de Guatemala
Redes CC3067
Kenneth Galvez 20079
Natanael Giron 20029

Discusión

A través de una metodología que involucró la implementación, pruebas y análisis, pudimos explorar los fenómenos que suceden en una comunicación de este estilo.

Con el programa de código de Hamming, se pudo evidenciar que cuando los cambios en los bits se producían de manera aleatoria, el rendimiento del código se veía significativamente afectado. En la mayoría de los casos, se lograron detectar errores, pero la capacidad para corregirlos se vio limitada, especialmente cuando se presentaban situaciones en las que más de dos bits se invertían o se cambiaban. También, en algunas instancias, se llegaba a observar que todos los bits sufrían modificaciones o cambios, cosa que el código de Hamming no podia corregir de manera exitosa. Estos cambios eran generados de forma aleatoria debido al cálculo de ruido implementado en el proceso, lo que en consecuencia provocaba que el código de Hamming detectara los errores. Por lo mismo, en ocasiones se presentaban resultados diferentes de los datos originalmente transmitidos desde el emisor desarrollado en Python.

Con el código de Fletcher, al realizar pruebas con un total de 10,000 mensajes, observamos una tasa de error promedio de 0.67, alrededor del 67% de los mensajes transmitidos dieron algún tipo de error que fue detectado por el receptor. Resaltar la importancia de los esquemas de detección implementados, ya que nos permitieron identificar y manejar la mayoría de los errores antes de que afectaran los mensajes finales. Creemos que esta tasa de error relativamente alta puede atribuirse en parte a la variabilidad en el tamaño del mensaje, los niveles de ruido y las probabilidades de error introducidas durante las pruebas.

Mientras que al reducir la cantidad de pruebas a 100, la tasa de error promedio disminuyó a 0.61. Lo que podría significar que una cantidad menor de pruebas resultó en una mayor precisión en la transmisión y recepción de mensajes. Además que observamos que la variabilidad en el tamaño del mensaje, el tiempo de procesamiento y el ruido aplicado también fue más significativo en comparación con las pruebas de mayor escala.

Comparando el Código de Hamming con Fletcher, nos dimos cuenta que al hablar de eficiencia, el código de Fletcher demostró ser mayor, ya que el trabajo de él era detectar los errores, cosa que hizo de buena manera, mientras que Hamming tuvo bastante dificultad a la hora de encontrar errores, porque si los detectaba pero no siempre eran correctos, ya que el código como que quebraba y no podria procesar

la cantidad de errores que tenia. Por lo mismo,pensamos que es mejor la implementación de Fletcher para la revisión de bits y detección de errores

Conclusiones

El código de Hamming se vió eficaz al momento de la detección de errores, pero no siempre concordaban o eran correctos los señalamientos, por lo que su capacidad es limitada bajo escenarios estresantes, como lo es cuando múltiples bits se modifican de manera simultánea.

Se facilita el trabajo para la detección de errores, a comparación de la corrección de los mismos, ya que los algoritmos suelen tener sus propias deficiencias al tener una sobrecarga de errores, por lo cual tienden a mostrar resultados diferentes a los esperados.

El código de Fletcher es un algoritmo sólido que a pesar de diferentes factores, como el tamaño del mensaje y el ruido, continúa teniendo una tasa de error adecuada siendo ideal cuando se lleva acabo este tipo de comunicación.

Citas y Referencias

El interfaz socket. (2023). Lcc.uma.es. http://www.lcc.uma.es/~eat/services/i_socket/i_socket.html