CC3067 - Redes
Sección 20
José Pablo Kiesling Lange - 21581
Erick Stiv Junior Guerra Muñoz - 21781

Laboratorio No.02 - Parte 2

Esquemas de detección y corrección de errores

Repositorio de GitHub del proyecto: TheKiesling/redes_lab2

Descripción y Metodología

En esta práctica se tuvo como objetivo establecer comunicación entre un emisor y un receptor, utilizando algoritmos de detección y corrección de errores. Los algoritmos implementados para este laboratorio fueron: Fletcher checksum y Código de Hamming, para la detección y corrección de errores, respectivamente.

Para establecer la comunicación se usaron las siguientes capas:

- Aplicación: Al correr el emisor se tiene que mandar por parámetro de CMD el mensaje a enviar y el receptor tiene que estar corriendo para poder recibir mensajes.
- Presentación: Una vez se recibe el mensaje, este se codifica en código binario. El receptor, al verificar la integridad del mensaje (y corregirlo si se pudiera y fuera necesario) lo decodifica.
- Enlace: El emisor usa cualquiera de los dos algoritmos para aplicar al mensaje codificado por la capa anterior. El receptor usa ese mismo algoritmo para detectar y corregir errores (según sea el caso y la necesidad), con el fin de verificar la integridad del mensaje.
- Ruido: Como tal no es una capa teórica, sin embargo, por motivos de la práctica se agregó para pruebas. Esta se basa en un porcentaje de ruido para ver si cambia un bit en el mensaje que recibe de la capa de enlace (en el caso de Hamming, puede ser también en los bits de paridad)
- Transmisión: Se usan sockets en ambos miembros para enviar y recibir mensajes. Para ello, se usa la librería de cada lenguaje según sea el caso.

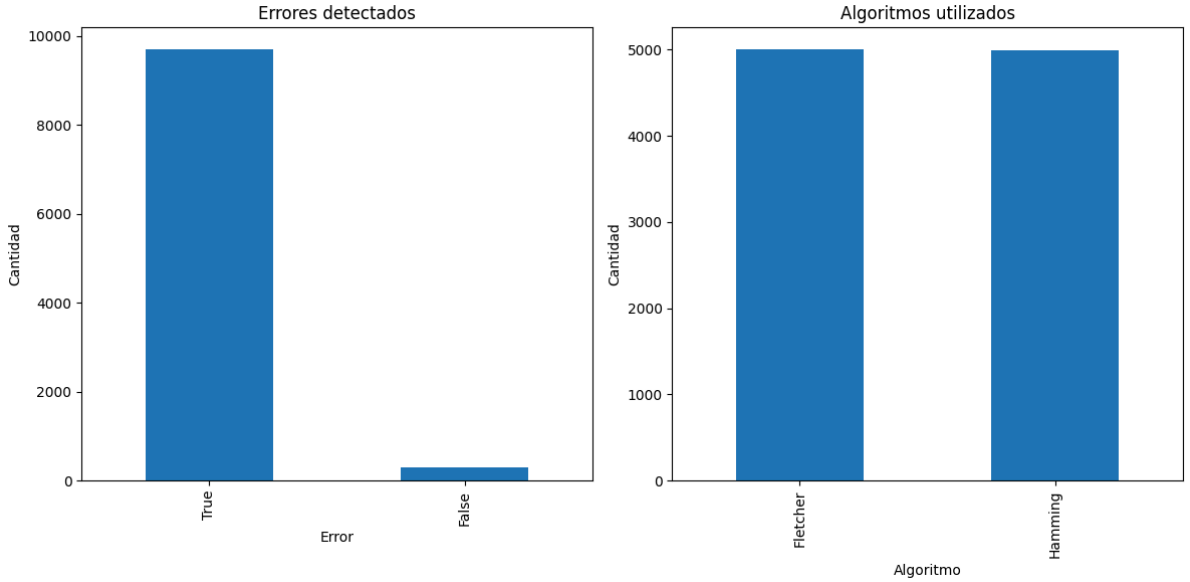
Para la parte de código, se unificaron los algoritmos en un solo directorio para que el emisor y el receptor tuvieran ambos a disposición. Además, como uno de los objetivos del laboratorio era crear un ambiente de prueba, se cambió el esquema de ejecución, en donde al correr el programa del emisor se debía pasar como parámetro por CMD el mensaje a enviar. En el nuevo esquema, el emisor crea automáticamente una cantidad especificada de mensajes de longitud aleatoria, con probabilidad de error aleatoria entre 0 y 1, de manera que se permitía la automatización de pruebas y generación de gráficas finales.

Como se menciona, para las pruebas se envían mensajes aleatorios de longitud aleatoria, con probabilidad de error entre 0 y 1. Se verifica el estado del error para que puedan ser detectados y/o corregidos según sea el caso.

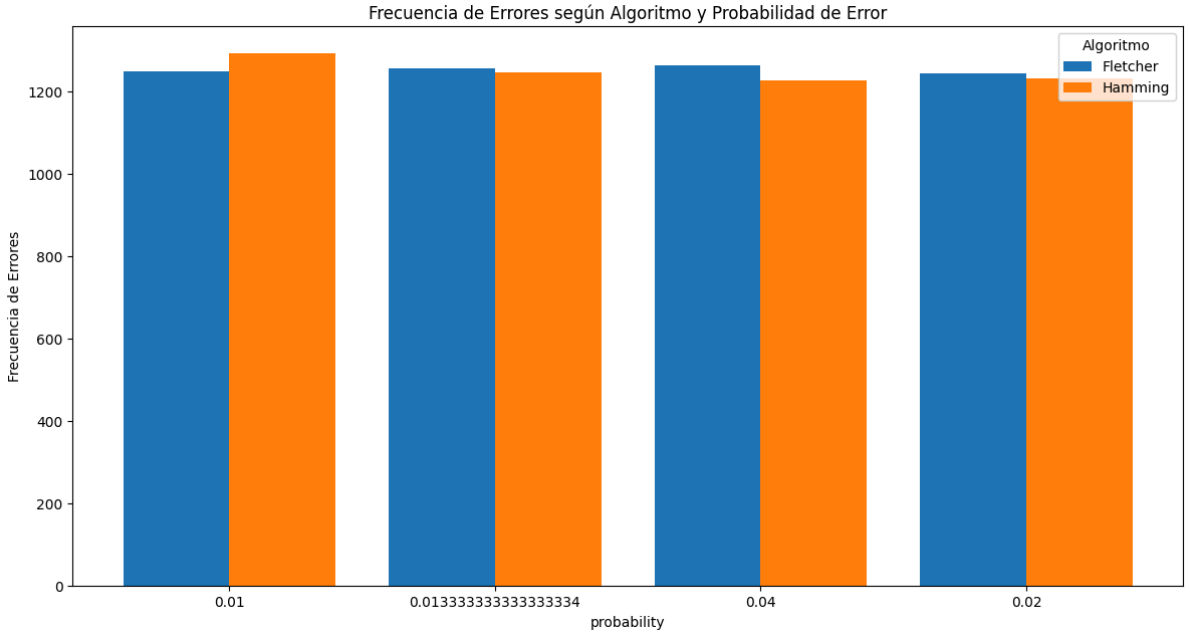
Resultados

Test con 10,000 iteraciones:

Imágen 1: Frecuencia de errores detectados según Ocurrencia y Algoritmo

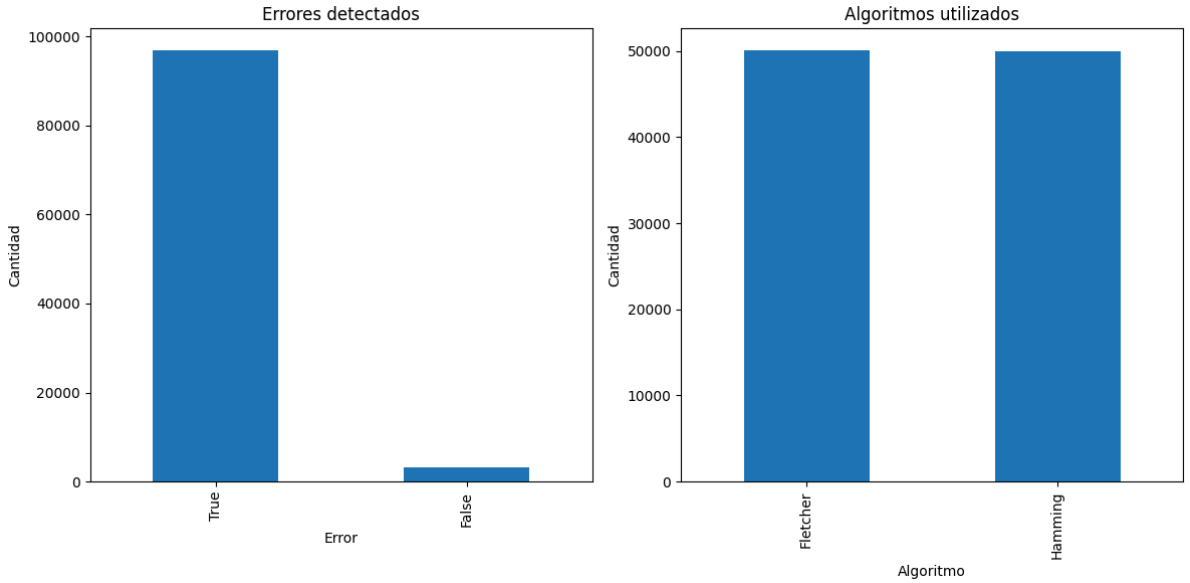


Imágen 2: Frecuencia de errores según Algoritmo y Probabilidad de Error

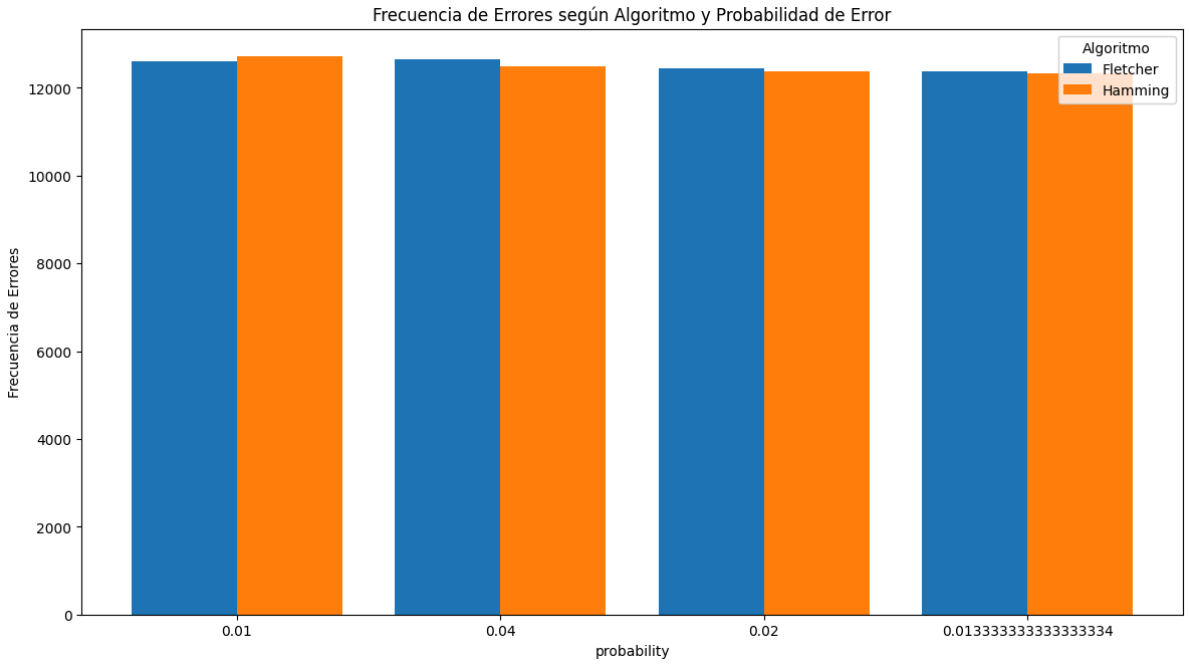


Test con 100,000 iteraciones:

Imágen 1: Frecuencia de errores detectados según Ocurrencia y Algoritmo



Imágen 2: Frecuencia de errores según Algoritmo y Probabilidad de Error



Discusión

El algoritmo de Hamming, como se sabe, cuenta con la capacidad no solo de detectar errores sino también corregirlos según la implementación. En contraste, el algoritmo de Fletcher Checksum solamente es capaz de detectar si existe un error, sin llegar a corregirlo. Por ello, en términos de funcionamiento, se determinó que el algoritmo de Hamming es el más adecuado en sistemas de comunicación. Además, en las pruebas realizadas, el desempeño que tuvo para detectar y corregir errores fue óptimo, lo que demuestra su potencial en aplicaciones reales. Sin embargo, en términos de eficiencia, este algoritmo es inferior a Checksum, puesto que este último es más rápido y sencillo de calcular, lo cual optimiza de mejor manera los recursos, teniendo también en las pruebas realizadas una tasa de efectividad alta para detectar errores.

En términos de flexibilidad, cabe destacar que si bien Hamming detecta y corrige errores, esto solo ocurre cuando hay un error, es decir que en escenarios ideales, la tasa de ruido es menor o igual a $\frac{1}{longitud\ mensaje\ en\ bits}$ ya que asegura que solo habrá un error y este será detectado. En escenarios con tasas más altas a la descrita anteriormente, el algoritmo no funciona correctamente y puede corregir de manera errónea, generando un mensaje que no tiene relación al enviado. Por otro lado, a pesar de que Checksum no corrija errores, como se vio en la primera parte de este laboratorio, sí es capaz de detectar errores múltiples en los mensajes, lo cual presenta una gran ventaja a la hora de que los errores escalen, en comparación a Hamming.

Los escenarios en donde se puede usar un algoritmo de detección y uno de corrección dependen de aspectos como hardware y software o los requerimientos del sistema. En el caso de Checksum, se puede utilizar si hay un límite en ancho de banda, como se vio en clase. Además, si se sabe que la transmisión de mensajes se da en un entorno con mucho ruido, se puede optar a usar más este algoritmo, ya que como se mencionó anteriormente, posee más flexibilidad con mayor cantidad de errores. Dicha detección incluso podría ser útil solo para saber si es confiable o no el mensaje y no el contenido de este. En contraste, Hamming se recomendaría más en escenarios donde los requisitos de la integridad del mensaje sean bastante críticos. Esto es porque, como se sabe, con este algoritmo se puede corregir el error que se integre a los mensajes.

En conclusión, Hamming es superior a la hora de funcionamiento como tal, ya que permite corregir errores. Sin embargo, en términos de rendimiento y flexibilidad de errores, Checksum es superior. No obstante, se recomienda estudiar debidamente el entorno y la aplicación que se desea para determinar el mejor algoritmo a emplear.

Conclusiones

- Se determinó que Hamming fue óptimo para corregir errores y garantizar la integridad del mensaje cuando llega un solo error en el mismo; mientras que Fletcher Checksum siempre detectó los errores sin importar la cantidad de estos.

- Se encontró que la velocidad de ejecución de Fletcher Checksum fue más eficaz en comparación con la de Hamming, lo cual puede ser de utilidad para sistemas con pocos recursos o en los que estos sean limitados.

- Se recomienda que, en sistemas con límites de recursos o entornos con mucho ruido, sea preciso utilizar un algoritmo de detección. Por otro lado, en sistemas donde sea prioritario procurar la integridad del mensaje, o bien, en donde el ruido sea mínimo, se recomienda optar por un algoritmo de corrección.