Universidad del Valle de Guatemala
18 Avenida 11-95 Guatemala
Facultad de Ingeniería
Departamento de Computación

Detección y corrección de errores - pt 2

Integrantes:
Diego Alexander Hernández Silvestre, 21270
Linda Inés Jiménez Vides, 21169

Curso:
Redes

Sección:
Sección 10

Guatemala, 1 de agosto de 2024

Repositorio GitHub: https://github.com/Dahernandezsilve/Lab2RedesP2.git

Descripción

En esta práctica de laboratorio se nos solicitó realizar una aplicación para la transmisión y recepción de mensajes, la cual va conectada a los algoritmos de detección y corrección de errores que se nos solicitó en el laboratorio pasado. Con ello se nos solicitaron varias pruebas para probar el funcionamiento de los algoritmos también aplicando un poco de ruido. En nuestro caso se realizaron un aproximado de 80 mil pruebas las cuales se presentarán más a detalle más adelante en el documento.

Metodología

Para esta práctica ya se contaban con los algoritmos de corrección y detección de errores, por lo que directamente se procedió a hacer la aplicación solicitada. Para ello se realizó un cliente en *python* y un servidor en *c++*. La forma en la que se desarrolló el proceso se basa en la descripción de servicios que se proporcionó como guía y se ve de la siguiente forma:

1. **Aplicación:**
   a. **Solicitar mensaje (Cliente):** En este proceso se realiza la generación n cantidad de mensajes mediante una funcion que genera mensajes aleatorios de 20 caracteres de extensión. Además, se le permite al usuario decidir qué algoritmo de integridad utilizar (Fletcher o Hamming). Sin embargo, para la automatización del proceso se realizaron pruebas con 20 mil mensajes para cada algoritmo.
   b. **Mostrar mensaje (Servidor):** En esta sección se gestiona la recepción de mensajes y verificación de errores. La lógica que se implementó permite la visualización de errores, detección de errores y correcciones si el algoritmo lo permite. En caso que el algoritmo no pueda corregir errores, los mensajes son descartados. Todas las acciones se presentan mediante impresiones de control dependiendo del caso.
2. **Presentación:**
   a. **Codificar mensaje (Cliente):** La forma en la que se realiza es solicitando un mensaje en ASCII y luego convirtiéndolo a su representación binaria para que las funciones que utilizan los algoritmos puedan interpretarlo correctamente y modificarlo de acuerdo al método seleccionado.
   b. **Decodificar mensaje (Servidor):** Se reciben los mensajes codificados en un formato *mensajecodificadoenbinario;algoritmo;tamañodebloque* y se utilizan los algoritmos configurados de acuerdo a lo recibido para decodificar el mensaje. Posteriormente, si se logra realizar alguna corrección o es recibido sin errores el mensaje, se vuelve a convertir a formato ASCII y se presenta el mensaje.
3. **Enlace:**
   a. **Calcular integridad:** Como se realizó en la primera parte del laboratorio, se modificaron ligeramente las funciones para ser utilizadas por el cliente y el servidor. En Fletcher se añade un checksum y en Hamming se añaden bits de paridad.
   b. **Verificar integridad (Servidor):** El servidor se encarga de validar el checksum y corregir errores cuando en el caso de utilizar Hamming, es posible.
4. **Ruido:**
   a. **Cliente:** A manera de simular interferencias durante la transmisión se introduce ruido con una probabilidad de error (adaptable) aplicada a cada bit incluyendo los bits de paridad en el caso de Hamming o checksum en el caso de Fletcher. La probabilidad que se decidió mantener para las pruebas fue de 0.001.

5. **Transmisión:**
   a. **Enviar información:** La transmisión se realiza mediante sockets TCP en Python. En total para la automatización de pruebas, se envían 80 mil mensajes en total.
   b. **Recibir información:** En el servidor se implementó un mecanismo que hace uso de hilos para manejar conexiones simultáneamente. De esta manera, se logra que cada cliente se conecte al servidor en un hilo separado sin que el servidor se bloquee al esperar nuevas conexiones. Además, en este mismo proceso, los resultados de las pruebas se van escribiendo y almacenando en un csv llamado Resultados.csv.

Resultados

Luego de realizar las 80 mil pruebas estas se exportaron a un csv el cuál facilitó el análisis de los datos con pandas y a su vez la realización de las gráficas. En la figura 2 podemos ver la comparación de los resultados de mensajes que fueron enviados con errores, sin errores a cada uno de los algoritmos para así un total de 80 mil mensajes. Estos se pueden ver más a detalle en la figura 3. La figura 1 nos muestra la cantidad de tipos de mensajes recibidos por porcentaje.

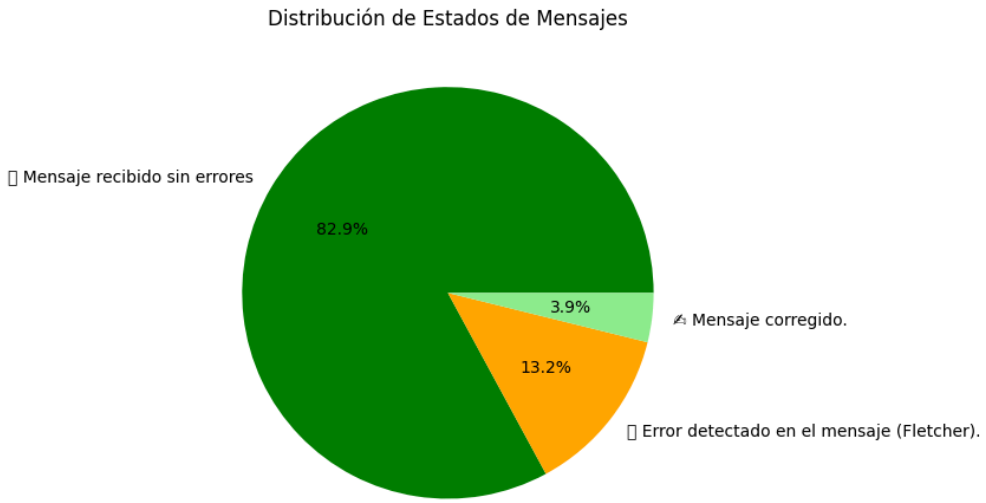

Figura 1. Porcentaje de distribución estado de mensajes

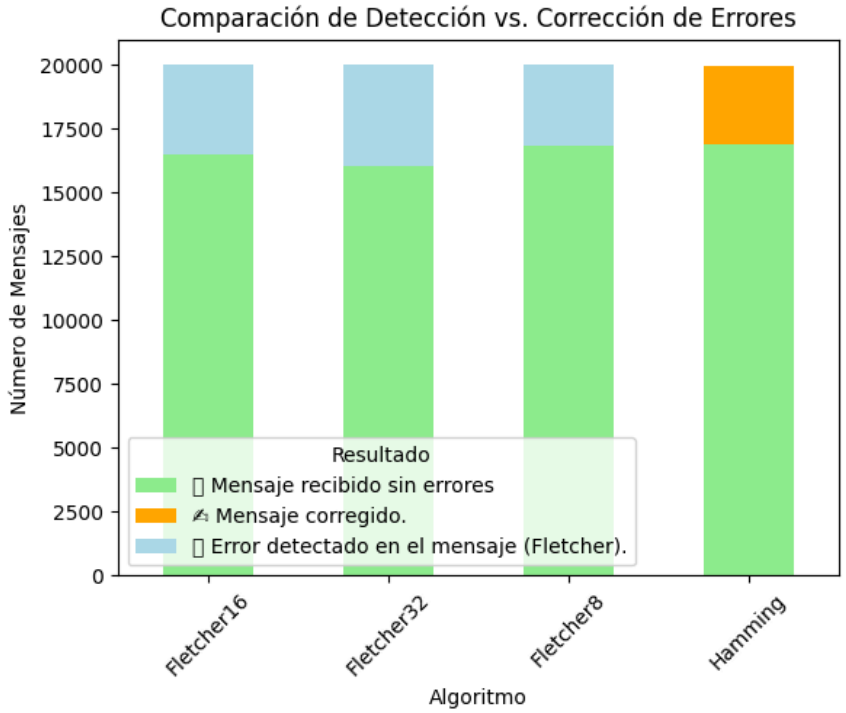

Figura 2. Mensajes detección y corrección de errores

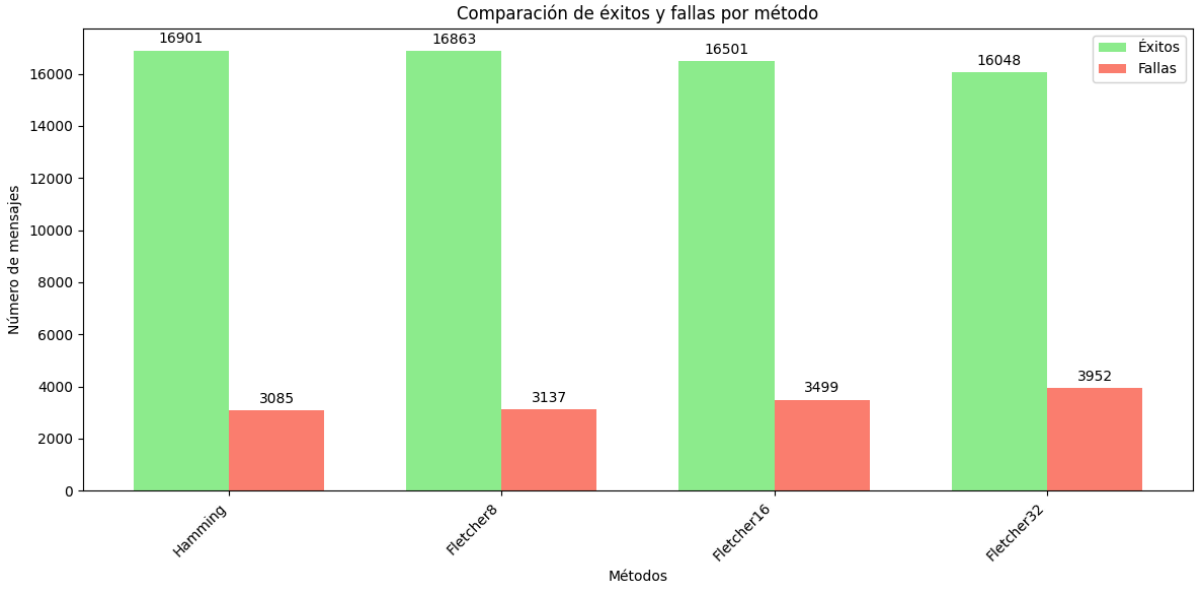

Figura 3. Mensajes exitosos y con fallos por cada método.

Luego se realizó unas gráficas de éxitos vs fallos de cada método utilizado para ver el comportamiento de cuantos mensajes con error se introducían entre los mensajes correctos.

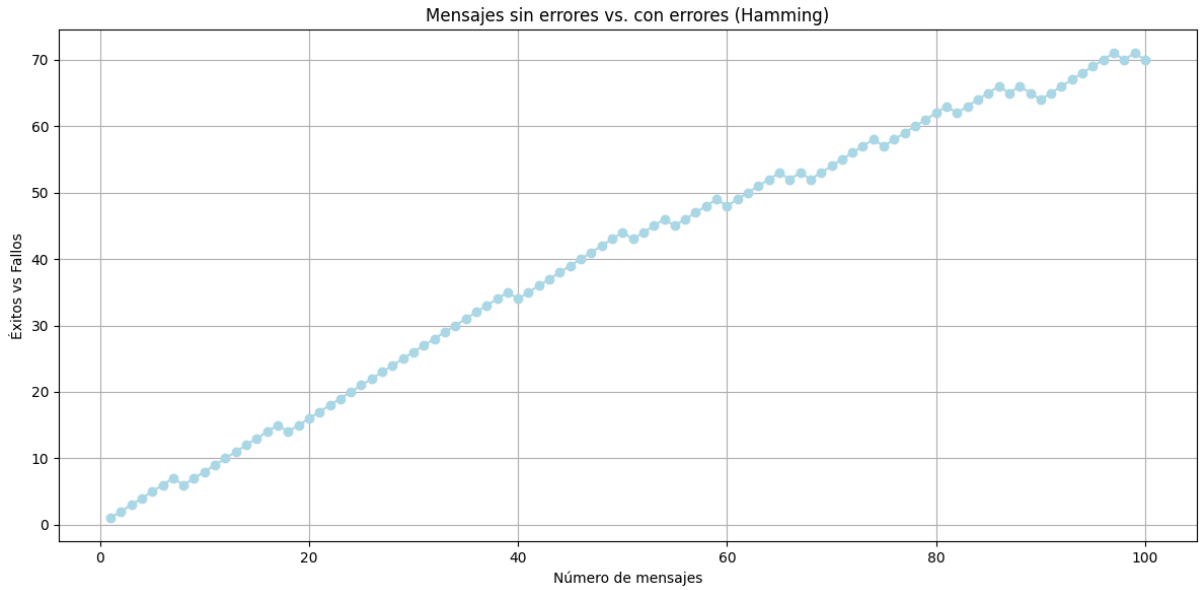

Figura 4. Éxitos y Fallos a lo largo del tiempo Hamming

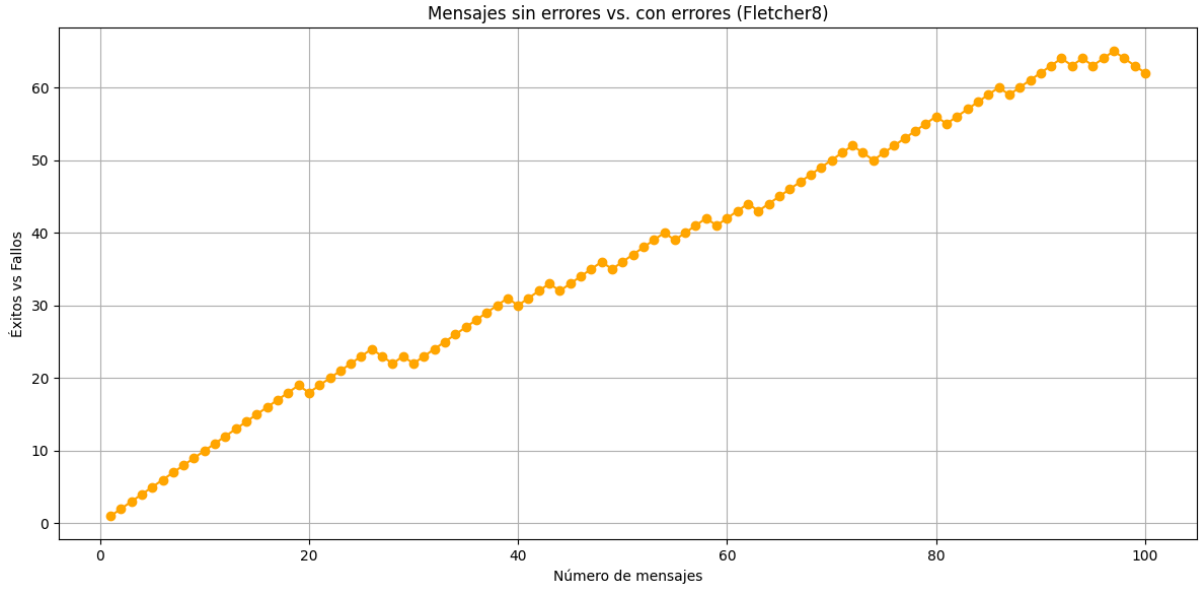

Figura 5. Éxitos y Fallos a lo largo del tiempo Fletcher 8

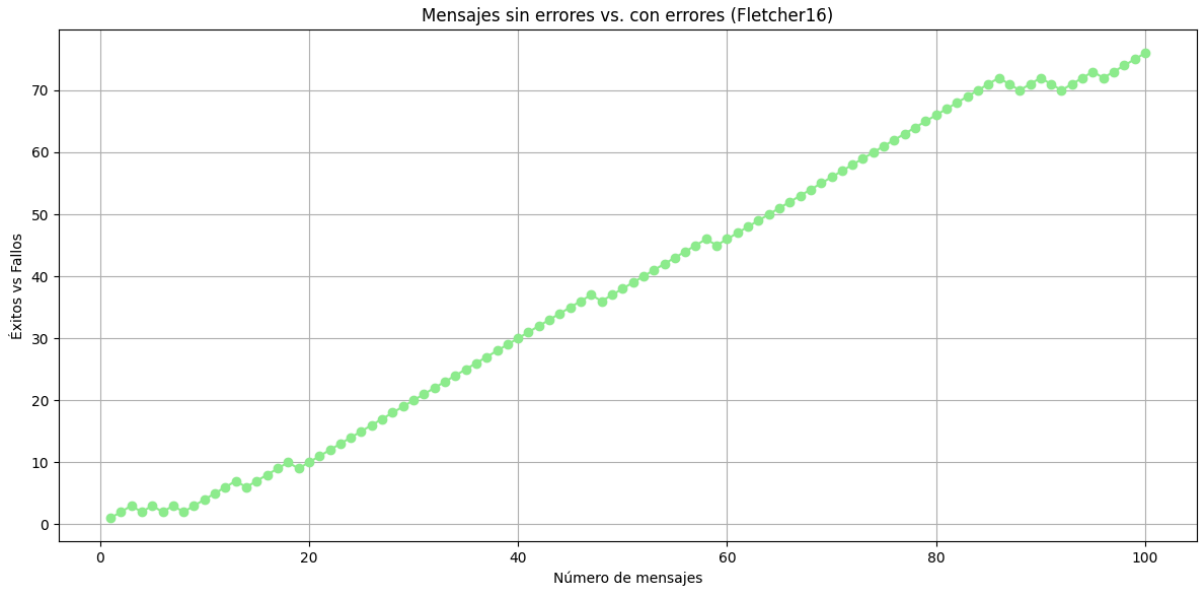

Figura 6. Éxitos y Fallos a lo largo del tiempo Fletcher16

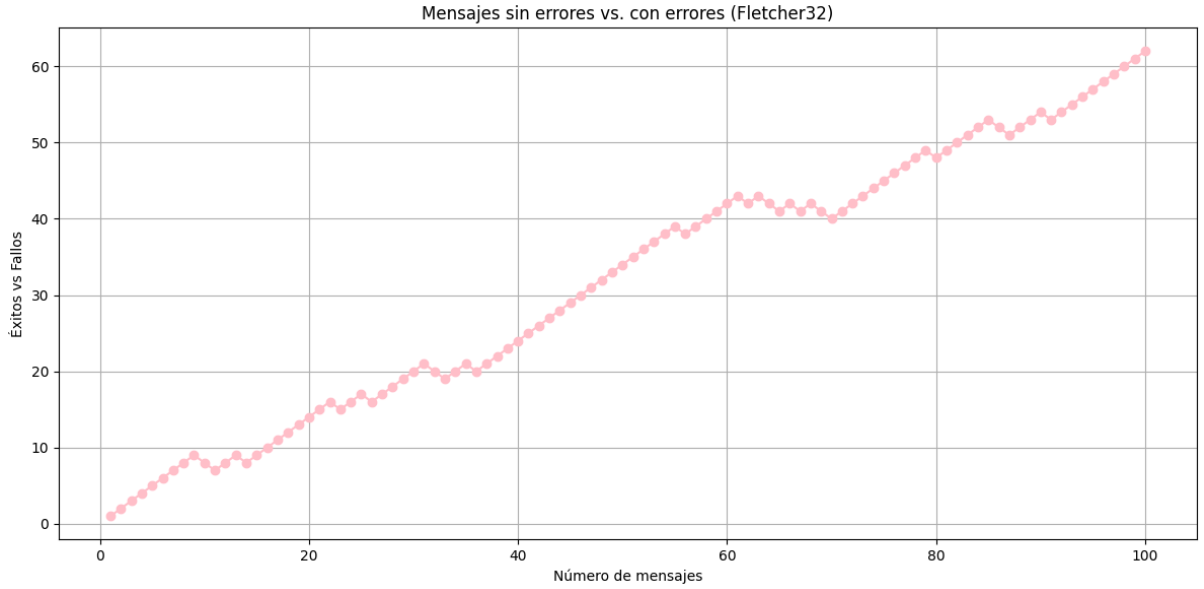

Figura 7. Éxitos y Fallos a lo largo del tiempo Fletcher32

Discusión

En la sección de resultados se mostraron algunos de los resultados obtenidos por las pruebas automatizadas de nuestros algoritmos que serían el de Hamming y el de Fletcher.

Como se mencionó anteriormente se realizaron 80 mil pruebas de las cuales como podemos ver en la figura 2 fueron 20 mil mensajes de prueba para cada algoritmo, dividiendo Fletcher en sus respectivos bloques de 8, 16 y 32 bits. Para ello se nos solicitó aplicar un poco de ruido a los mensajes para así ver el funcionamiento de nuestros algoritmos a la hora de detectar y corregir un error. La figura 1 nos muestra que el 82.9% de los mensajes iban correctos, del el 17.1% restantes el 13.2% fueron errores detectados por parte del algoritmo de Fletcher y el 3.9% fueron corregidos por el algoritmo de Hamming. Estos resultados fueron considerando que el ruido tenía una probabilidad de error de 0.001.

De la figura 4 a la 7 podemos apreciar la cantidad de éxitos y fallos de mensajes en cada uno de los algoritmos conforme se van ingresando mensajes. Podemos notar que en casi todos los algoritmos se ve una tendencia donde casi que la mayoría de mensajes son exitosos con ciertos altibajos por los fallos, aunque cabe destacar que el de Fletcher16 es el que va aumentando en su mayoría con mensajes exitosos con pocos fallos encontrados. Con estos resultados y también observando los de la figura 3 podemos ver que todos los algoritmos son bastantes similares en su detección o corrección de errores, aunque tomando en cuenta todos los de fletcher podemos ver que la mayoría fueron detectados más no corregidos.

Esto puede decirnos que al unir las 3 subdivisiones de Fletcher este pudo tener un mejor rendimiento al tener mayor capacidad al momento de realizar el envío y detección errores a comparación de Hamming que este sí los corrige. Con ello se puede decir que Fletcher es capaz de detectar la mayoría de errores presentes y es más eficaz al momento de detectar más de un error debido a que, en el caso de Hamming, si existe más de un error en el mensaje, la corrección no será adecuada ya que es capaz de corregir un solo error. Cabe destacar, que el algoritmo de Hamming tiene la ventaja de poder corregir errores y se considera más beneficioso que Fletcher dado que tiene más valor un mensaje que se intentó corregir que al otro que se descartó por detectar errores.

Por estas razones, es recomendable utilizar un algoritmo de corrección cuando queremos garantizar que el envío de mensajes es continuo y nos interesa que la red se mantenga con la menor pérdida como en un formato de streaming de video, audio, etc. Por otra parte, cuando se prioriza calidad de información debido a que una pequeña variación puede provocar grandes problemas, es mejor utilizar un algoritmo de detección de errores para comprobar la integridad del mensaje y evitar problemas que pueden llevar a estudios mal hechos. En estos últimos, se podrían encontrar experimentos científicos, manejo de información bancaria, entre otros. *(Rastersoft, s.f).*

Con ello se puede concluir que ambos algoritmos tienen sus ventajas y desventajas en situaciones distintas. Un algoritmo de corrección de errores tiene ventaja sobre uno de corrección cuando el objetivo es solo ir detectando errores rápidamente y no se sabe que tipo de errores pueda contener este debido a las limitaciones que puedan tener los de corrección de errores.

Conclusiones

- Dado el porcentaje de errores enviados a los algoritmos se notó que fletcher obtuvo un mejor rendimiento al detectar la mayor parte de los errores enviados.
- A pesar de que los algoritmos como Hamming puedan corregir errores, los de detección llevan la ventaja de soportar más errores en algún mensaje corrupto.
- La incidencia de errores se ve reflejada por el porcentaje de ruido que fue implementado.
- La implementación de hilos redujo significativamente el tiempo de la automatización para las pruebas en envío y recepción.
- Los algoritmos de detección de errores se pueden utilizar cuando se prioriza la calidad de la información.
- Los algoritmos de corrección de errores se pueden utilizar cuando se prioriza mantener una comunicación estable en todo momento.

Referencias

Rastersoft (s/f). *Detección y corrección de errores*. Recuperado el 1 de agosto de 2024, de https://rastersoft.com/articulos/errores.html

Fastercapital. (15 de junio de 2024). Recuperado el 1 de agosto de 2024, de https://fastercapital.com/es/contenido/El-papel-de-CRC-en-la-deteccion-de-errores--asegurar-la-integridad-de-los-datos.html