CC3067 - Redes
Sección 10

Pablo Andrés Zamora Vásquez - 21780 Diego Andrés Morales Aquino - 21762

Laboratorio No. 2
Esquemas de detección y corrección de errores - Parte 2

Link al repositorio de GitHub:

https://github.com/pabloozamora/UVG_Redes_Lab_2/tree/pt_2

Descripción

Esta práctica de laboratorio consistió en implementar una arquitectura de capas para la transmisión de mensajes, en la cual, el emisor fue desarrollado en Python y el receptor, en Javascript. Esta implementación permite al usuario ingresar un mensaje para enviar desde el emisor al receptor, así como decidir el algoritmo que se utilizará para la validación o corrección del mismo.

Una vez finalizada la implementación, se realizó un escenario de pruebas en el que se enviaron 10,000 mensajes, variando la longitud del mensaje, la probabilidad de error por bit y el algoritmo a utilizar en la codificación y decodificación. A partir de los resultados de este, fue posible identificar las condiciones que influyen en el rendimiento de cada algoritmo.

Metodología

Para realizar la arquitectura planteada, se implementaron las siguientes capas, en las cuales, el emisor utiliza Python como lenguaje de programación, mientras que el receptor utiliza JavaScript:

Capa de aplicación

Al realizar el envío y recepción de un solo mensaje, la capa de aplicación consistió en solicitar al usuario emisor por medio de la consola, el mensaje a enviar y el algoritmo a utilizar. En el caso del receptor, también se le solicita al usuario que especifique el algoritmo a utilizar. Luego del envío, codificación y decodificación del mensaje, también se muestra en pantalla si el mensaje recibido es correcto o posee errores. En el caso de ser correcto, también muestra el mensaje original enviado por el emisor.

Al efectuar las pruebas automatizadas, la capa de aplicación generó de forma aleatoria los mensajes a enviar, realizando 10,000 iteraciones de prueba con ambos algoritmos.

Capa de presentación

En la capa de presentación se realizaron los servicios de codificar una cadena de caracteres a código binario, utilizando el código ASCII para representar cada caracter como una cadena de bits. Así también, se realizó el proceso inverso de decodificación. La codificación y decodificación se emplearon para los emisores y receptores respectivamente.

Capa de enlace

En la capa de enlace, dependiendo del algoritmo seleccionado por el emisor, del lado del emisor se calcula la información necesaria para asegurar la integridad del mensaje. En el caso del algoritmo de verificación de Fletcher, se calcula el checksum de la cadena de bits y se concatena al final de esta; mientras que, con el algoritmo de corrección de Hamming, se realiza el cálculo y se añaden los respectivos bits de paridad en sus posiciones correspondientes.

Por otro lado, en el caso del receptor, utilizando el algoritmo correspondiente, se realiza la verificación de la integridad del mensaje. En el caso del algoritmo de Fletcher, se recalcula y compara el checksum; en cambio, en el caso del algoritmo de Hamming, se examinan los bits de paridad y dependiendo de si estos indican un error, se procede a realizar la corrección del mensaje.

Capa de ruido

Para simular el ruido del medio al momento de la transferencia de mensajes, se integró una capa adicional, la cual se encarga de cambiar un bit determinado según una probabilidad de error dada. Para cada bit en el mensaje, se genera un número aleatorio entre 0 y 1; si este número es menor que la probabilidad de error especificada, el bit se invierte, simulando así el efecto del ruido en el medio de transmisión. El resultado es un nuevo mensaje con posibles errores. Cabe destacar que el ruido se aplicó tanto al mensaje original en binario como a los bits de verificación o de paridad.

Capa de transmisión

Para simular la transmisión de datos, se implementaron *sockets*. En el lado del receptor; es decir, el servidor de Javascript, el socket escucha constantemente los mensajes que pueda recibir de cualquier cliente. Por otro lado, en el emisor; es decir, el cliente de Python, el socket se conecta al servidor para poder enviar mensajes.

Escenarios de pruebas

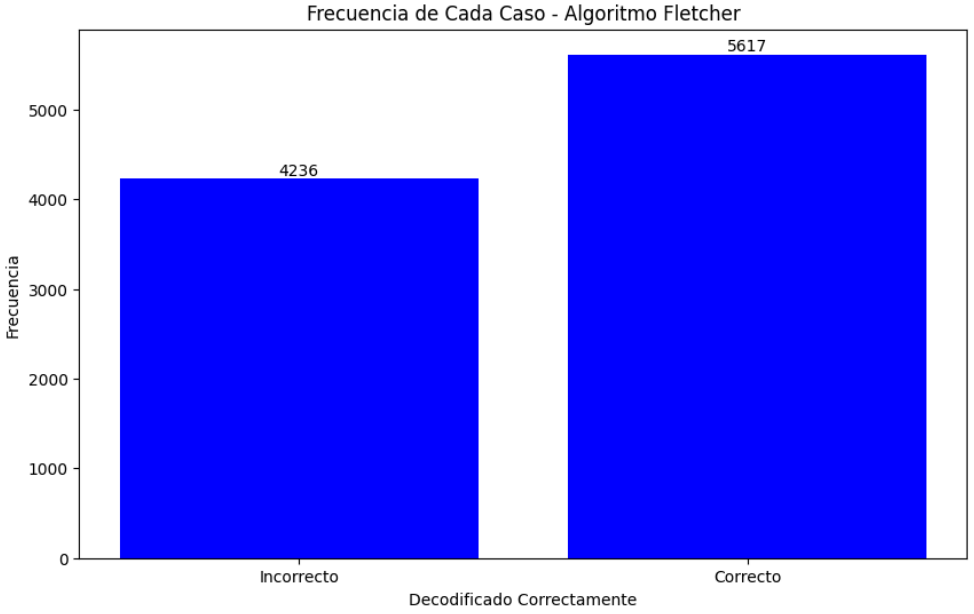
Para poner a prueba la efectividad de ambos algoritmos, se modificaron ligeramente el servidor y el cliente.

En primer lugar, únicamente con fines de prueba, se definió que los mensajes que el emisor transmite al receptor deben incluir cierta “metadata” en un objeto JSON. En este, se especifica la probabilidad de error para cada bit, el mensaje original y el algoritmo a utilizar para validar y/o corregirlo. Esto se realizó con dos propósitos: primero, hacer posible la identificación de aquellos mensajes con más de un bit de error en los que el algoritmo de Hamming corrige erróneamente el mensaje; y segundo, hacer posible el registro del algoritmo y la probabilidad de error por bit para cada mensaje (puesto que estos parámetros varían durante las pruebas).

Además, se automatizó el cliente de manera que envíe un número determinado de mensajes por sí solo.

Resultados

Al enviar 10,000 mensajes de prueba, variando el algoritmo de validación/corrección, la probabilidad de error por bit y la longitud del mensaje, fue posible generar un archivo CSV que permitió obtener la cantidad de mensajes que fueron posibles decodificar con cada algoritmo, tal como se muestra en la imagen 1.



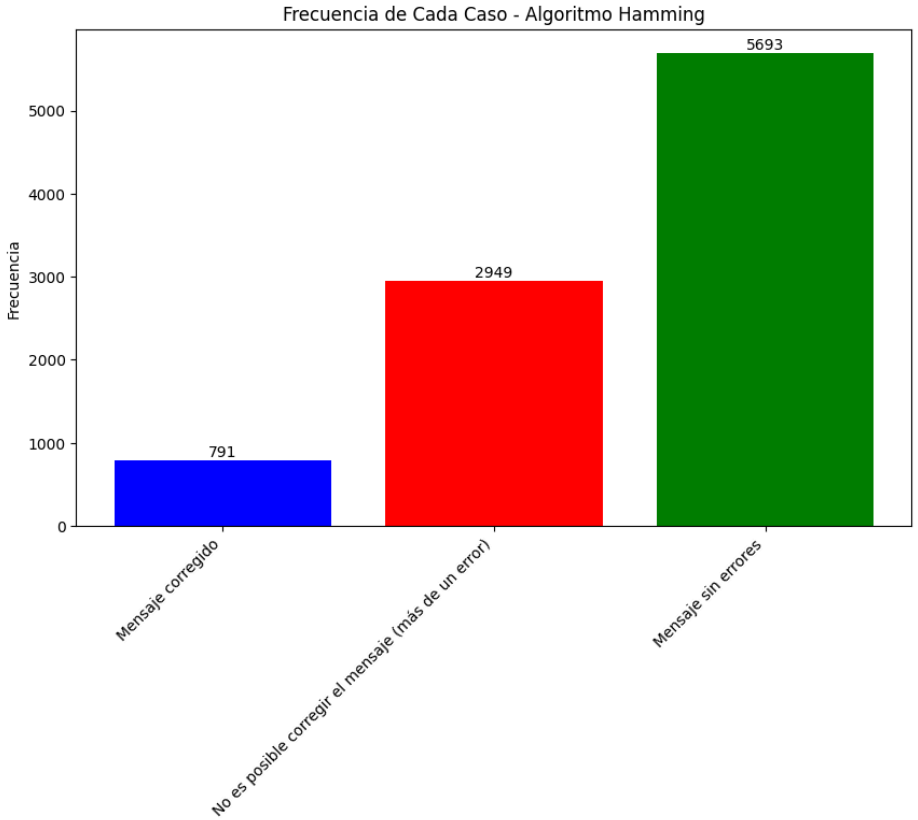


Imagen 1. Frecuencia de casos para cada algoritmo al enviar 10,000 mensajes con cada uno.

En estas gráficas es posible observar que, utilizando el algoritmo de Fletcher, únicamente fue posible decodificar 5617 mensajes, aquellos que no presentaron errores; mientras que, utilizando el algoritmo de Hamming, adicional a los 5693 mensajes sin errores, fue posible decodificar 791 mensajes que únicamente presentaban un error.

Adicionalmente, fue posible agrupar los resultados de cada algoritmo según la tasa de errores para así identificar su comportamiento conforme esta crecía, tal como se muestra en la imagen 2.

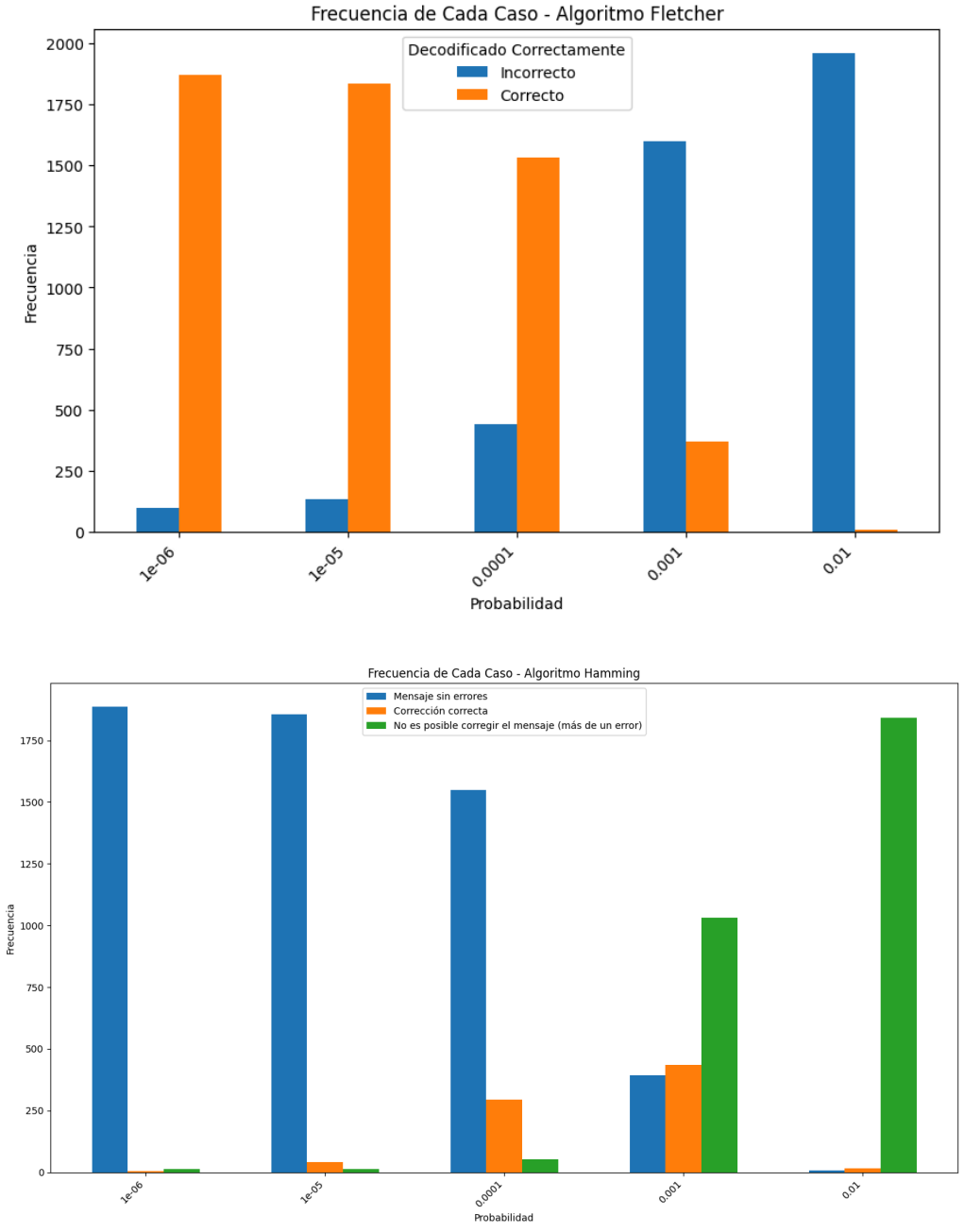


Imagen 2. Frecuencia de casos para cada algoritmo al enviar 10,000 mensajes con cada uno, agrupados por probabilidad de error.

Puede notarse que, con ambos algoritmos, la cantidad de mensajes posibles de descifrar posee una tendencia a disminuir a medida que la tasa de errores en los mensajes aumenta.

Por otro lado, para cada uno de los algoritmos, se obtuvo la frecuencia de mensajes que presentaron algún error y se agrupó según la longitud del mensaje original enviado. Tal y como se observa en la imagen 3, en un inicio, el algoritmo de Fletcher presentó una diferencia significativa en los mensajes con error detectados. Sin embargo, dicha diferencia entre ambos algoritmos disminuyó a medida que se incrementó la longitud de los mensajes enviados.

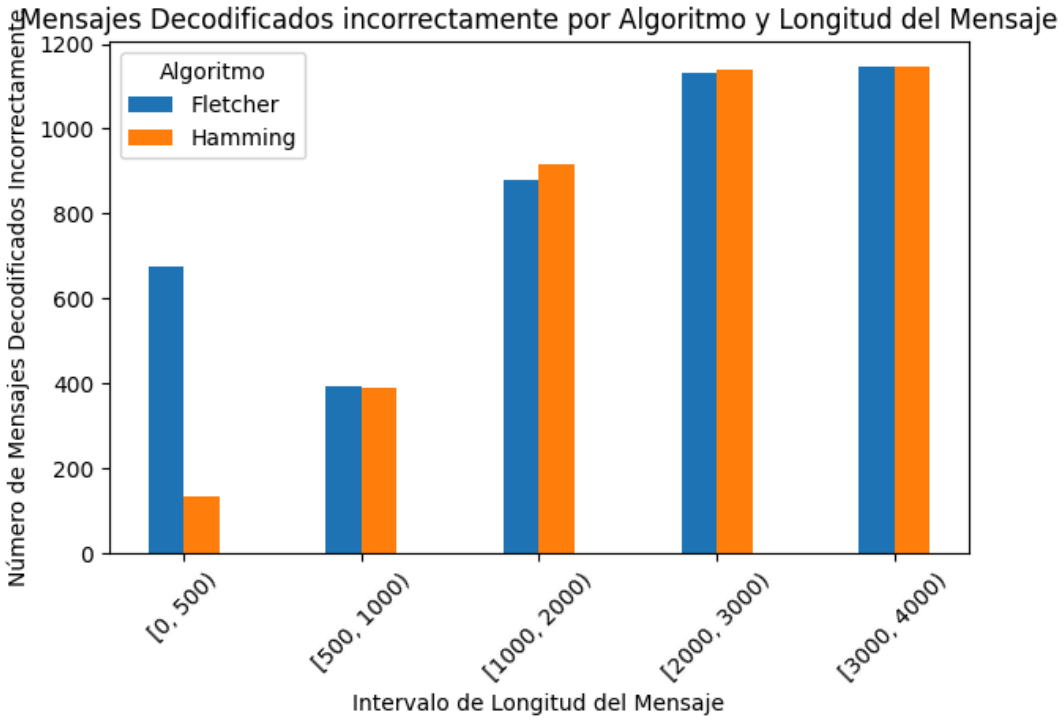


Imagen 3. Frecuencia de casos incorrectos para cada algoritmo al enviar 10,000 mensajes con cada uno, agrupados por longitud del mensaje original.

Discusión

Al observar los resultados obtenidos, el algoritmo de Hamming podría aparentar ser el algoritmo con mayor efectividad, pues al incluir los mensajes con error que fueron corregidos, se logró obtener una mayor cantidad de mensajes que mantuvieron su integridad durante la comunicación. Sin embargo, es importante tomar en cuenta que durante dicha corrección, la cantidad de mensajes corregidos de forma incorrecta (es decir, mensajes con más de un error), supera por mucho la cantidad de correcciones exitosas. Por tanto, no es posible confiar completamente en dicho algoritmo de corrección, pues, a diferencia del presente ejercicio, en ambientes reales no se cuenta con la suficiente información para determinar si una corrección mantiene el contenido original transmitido por el emisor del mensaje.

Por otro lado, aunque el algoritmo de Fletcher no resulta tan eficiente como el de Hamming al emplear mensajes pequeños, la imagen 2 indica que, al aumentar la probabilidad de error por bit, la capacidad de corregir un solo error del algoritmo de Hamming ya no supone una

ventaja considerable sobre el de Fletcher en cuanto a la decodificación correcta de un mensaje, por lo que es una condición a tomar en cuenta al momento de elegir entre un algoritmo u otro en la transmisión de mensajes.

Adicionalmente, la imagen 2 también indica, como era de esperar, que ambos algoritmos disminuyen considerablemente la cantidad de mensajes que decodifican correctamente conforme aumenta la probabilidad de error por bit.

Al descartar la funcionalidad de corrección y tratando a ambos algoritmos como algoritmos exclusivos de verificación, se puede observar un mejor rendimiento por parte del algoritmo de Hamming. Tal y como se muestra en la imagen 3, los mensajes recibidos con uno o más errores son mucho más frecuentes en cadenas pequeñas al emplear el algoritmo de Fletcher. Esto se debe a la diferencia en el tamaño de la cadena de bits obtenido una vez codificado el mensaje con cada algoritmo. Por un lado, el algoritmo de Hamming calcula la cantidad de bits de paridad necesarios en función del tamaño de la cadena, mientras que, el checksum empleado por el algoritmo de Fletcher mantiene su tamaño fijo de 16 bits sin importar el tamaño de la cadena. Es decir que, para cadenas pequeñas, el algoritmo de Hamming añade una menor cantidad de “bits de verificación”, lo cual se traduce en una cadena de bits más pequeña a enviar y, por lo tanto, genera una menor probabilidad de que alguno de sus bits se voltee.

Tomando en cuenta lo anterior, el algoritmo de Fletcher resultaría más efectivo para cadenas muy largas, específicamente cadenas superiores a 65,519 bits de datos, con lo cual se necesitarían 17 o más bits de paridad en Hamming. Sin embargo, la longitud de la cadena sería tan grande que dicho factor sería despreciable.

Finalmente, En base a las condiciones que afectan el rendimiento de cada algoritmo mencionadas anteriormente, es posible afirmar que el uso de un algoritmo de detección es más adecuado en situaciones donde la comunicación es generalmente confiable y la frecuencia de errores es baja, o cuando es crucial garantizar la integridad del mensaje, permitiendo así la retransmisión de mensajes con errores. Por otro lado, en entornos con alta probabilidad de errores, donde la retransmisión es poco viable o demasiado costosa y se puede tolerar el riesgo de corrupción del mensaje, los algoritmos de corrección suelen ser una mejor opción al corregir el error sin necesidad de volver a enviar el mensaje.

Comentario

La práctica en cuestión permitió comprender más a profundidad el funcionamiento de algoritmos de detección y corrección de errores mediante la simulación de un escenario de intercambio constante de mensajes. Gracias a esto, fue posible identificar las condiciones que afectan el rendimiento de cada uno, resaltando así la importancia de elegir uno u otro según la situación en la que se desean implementar.

Conclusiones

- Para ambos algoritmos, al aumentar la probabilidad de error por bit, la cantidad de mensajes posibles de decodificar tiende a disminuir.

- El algoritmo de Hamming resulta más eficiente que el de Fletcher al decodificar mensajes correctamente cuando hay una baja probabilidad de error por bit, gracias a

su capacidad de corregir un solo error en el mensaje; sin embargo, cuando esta probabilidad aumenta, la cantidad de correcciones erróneas supera considerablemente la cantidad de correcciones exitosas.

- En cuanto a la validación de mensajes, el algoritmo de Hamming también resulta más eficiente que el de Fletcher, puesto que este únicamente únicamente agrega al mensaje original la cantidad de bits de paridad necesarios según su longitud; mientras que el de Fletcher siempre agrega 16 bits para el *checksum*, haciéndolo más propenso a errores.