Universidad del Valle de Guatemala
Facultad de Ingeniería

Redes
Laboratorio #2

Esquemas de detección y corrección de errores

Alejandro José Gómez Hernández 20347
Juan Carlos Baján Castro 20109

Descripción de la práctica y metodología utilizada:

La práctica presentada tiene como principal objetivo entender y analizar la importancia de los esquemas de detección y corrección de errores en la transmisión de información, especialmente cuando se está enfrentando un canal de comunicación propenso a interferencias. Este análisis es esencial para garantizar la integridad y confiabilidad de los datos enviados y recibidos en cualquier red de comunicación.

En cuanto a la metodología de la práctica, cabe destacar que esta se llevó a cabo en varias etapas. Primero, se analizó las funciones específicas de cada capa en la arquitectura de transmisión y recepción de mensajes brindadas en el archivo pdf. Después de analizar estos conceptos se inició con la implementación del emisor y receptor de cada algoritmo tomando como base la primera parte del laboratorio. Se aseguró la integración adecuada de los algoritmos de detección y corrección de errores.

Para validar la efectividad de estos algoritmos, se realizaron múltiples pruebas (10 mil pruebas) variando diferentes parámetros como el tamaño de las cadenas enviadas, la probabilidad de error, el algoritmo utilizado y la redundancia/tasa de código. Estas pruebas se automatizaron para permitir un análisis estadístico exhaustivo y la generación de gráficas que ilustran los resultados obtenidos. Se utilizó python y javascript para realizar el receptor y emisor de cada algoritmo. Para las tareas de estadísticas y análisis, se usó nuevamente python, en un jupyter notebook. Esto nos ayudó a visualizar de mejor forma los gráficos que se generaron.

Resultados

A continuación se presentan los resultados obtenidos por medio de la experimentación:

Algoritmo Hamming (7,4):

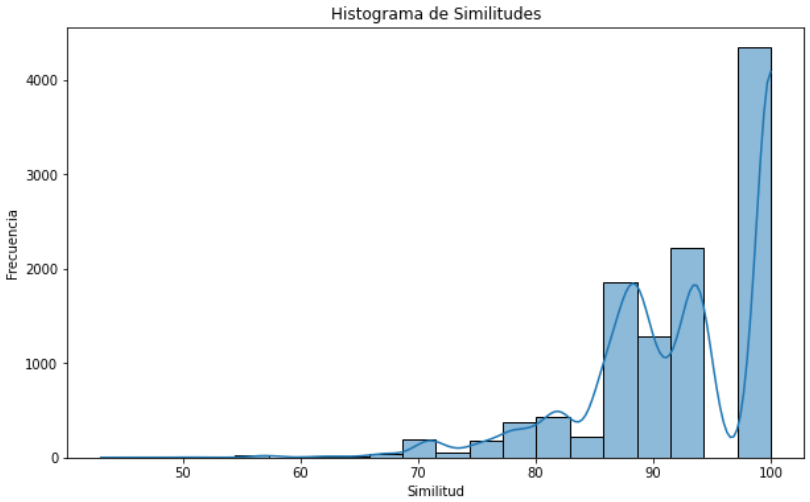
Muestra a utilizar:

Esta muestra se itera 750 veces con el objetivo de formar un total de 11249 palabras.
[Hola mundo, q dicen, ando bien, un poco estresado, pq hay q hacer, muchas cosas UVG, cosas del trabajo, cosas de musica, cosas de politica, pero todo se puede, familia, me tiembla el, ojo x ratos, es normal, no se xd,]

Estadísticas Descriptivas:

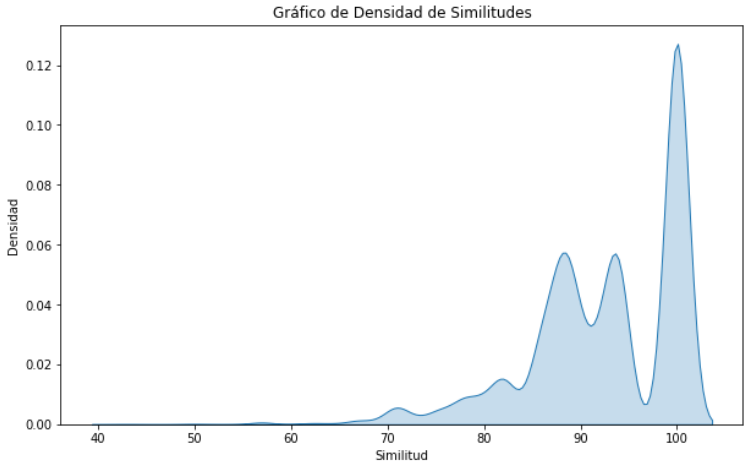
Similitud	
Count	11250
Mean	92.411644
Std	7.811375
Min	43
25%	88
50%	94
75%	100
Max	100
Mediana	94
Moda	100

Gráfico No.1: Histograma de similitudes para algoritmo Hamming



El histograma permite que se visualice de manera directa y discreta la cantidad de transmisiones que tienen un determinado número de errores.

Gráfico No.2: Gráfico de densidad para algoritmo Hamming



El gráfico de densidad proporciona una representación suavizada de la frecuencia con la que ciertas cantidades de errores (en nuestro caso, similitudes en Hamming) ocurren en el conjunto de datos generado. De esta forma, se pueden identificar con mayor facilidad las áreas donde los errores son más o menos frecuentes, lo que facilita la interpretación de la fiabilidad del algoritmo en la transmisión.

Con una muestra de 11,250 transmisiones, se ha observado que en promedio, hay una similitud grande del 92.41% entre los datos transmitidos y recibidos, lo que indica que el algoritmo ha sido efectivo en la mayoría de los casos. Sin embargo, la desviación estándar de 7.81 sugiere una variabilidad en los resultados. Esta variabilidad se confirma con un valor mínimo de similitud de tan solo 43%, lo que sugiere que en algunas ocasiones, el algoritmo no pudo corregir los errores con eficacia. Este fallo

podría deberse a una alta tasa de errores en la transmisión o a las limitaciones que pueda tener la implementación del algoritmo de Hamming cuando se encuentra con más de un error.

A pesar de estas variaciones, la mediana y la moda de los datos son 94 y 100, respectivamente. Esto implica que la mayoría de las transmisiones tuvieron una precisión bastante buena, con al menos la mitad de ellas alcanzando o superando una similitud del 94%. Incluso, la moda, siendo 100, destaca debido a que la similitud perfecta fue el resultado más comúnmente observado. La data obtenida nos dice que el 75% de las transmisiones lograron una similitud de un mínimo de 88%. Estos datos nos ayudan a entender que, aunque la mayoría de las transmisiones si llegan a ser fiables, siempre existe un margen significativo de transmisiones que podrían beneficiarse de otro tipo de algoritmos o de medidas adicionales para la corrección de errores.

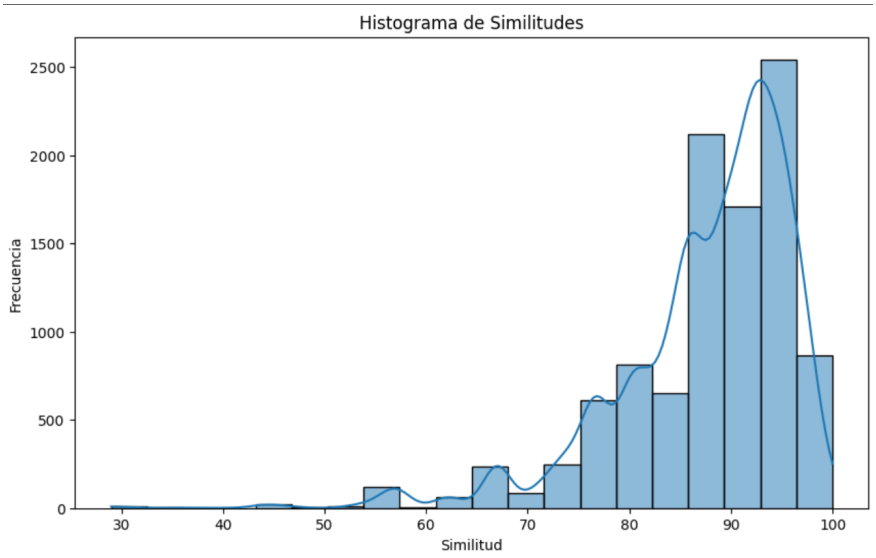
Algoritmo Fletcher Ketchum:

Muestra a utilizar:

Esta muestra se itera 440 veces con el objetivo de formar un total de 10,120 palabras.
[Hola Muy, Buenas Tardes, Como se, EnCuEnTrA, l DIA, de hoy, sabe, que hoy, se me, perdio un audifono, en la universidad, y tengo miedo de que, no lo pueda encontrar, tengo que decir, que ya llevaba mucho, tiempo con, ese, audifono y, era excelente, pero bueno, que puedo hacer, ahora debo buscarlo manana, porque hoy no ire :(E]

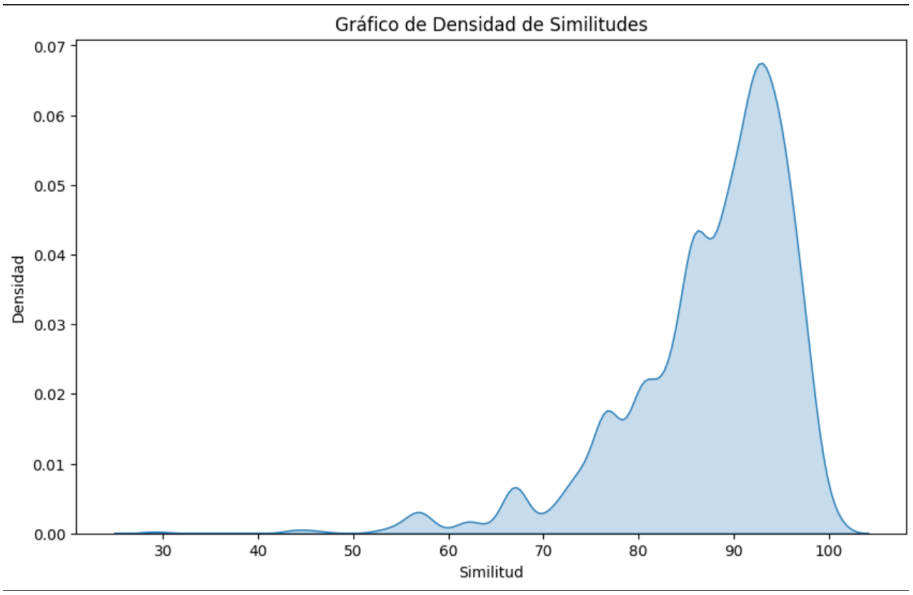
Para poder analizar el accuracy con la que el modelo envía las tramas se diseñó una función cuyo propósito devuelve la similitud entre la palabra original y la palabra codificada recibida en el receptor. Resultando en lo siguiente:

Gráfico No.3: Histograma de similitudes para algoritmo Fletcher Checksum



Como se puede observar la distribución se encuentra concentrada principalmente entre 87% y 100% indicando que el modelo acepta gran cantidad de errores al momento de verificar la integridad de el mensaje.

Gráfico No.4: Gráfico de densidad para algoritmo Fletcher Checksum



En esta gráfica se puede observar la densidad de las similitudes respaldando las observaciones mencionadas anteriormente. En términos generales y estadísticos se podría decir que el modelo es bastante bueno pues la media de diferencia no supera el 20%. Sin embargo, tomando en cuenta que hablamos de redes, entendemos que la comunicación debe ser siempre íntegra por lo que el modelo se vuelve bastante malo bajo esta perspectiva.

Estadísticas Descriptivas:

Similitud	
Count	10120
Mean	87.489723
Std	8.801480
Min	29.00000
25%	84.00000
50%	90%
75%	93%
Max	100.00000
Mediana	90.0000
Moda	86.0000

Con estos resultados podemos observar que el 50% de la distribución de los resultados posee un accuracy del 90%, hay un 25% concentrado solo entre 90%-93% y un 25% entre 0% - 84%. Reafirmando lo que se ha mencionado anteriormente, el accuracy es muy alto y estadísticamente aceptable. Sin embargo, para el contexto del que se habla es muy bajo.

Discusión:

El código de Hamming es uno de los esquemas de detección y corrección de errores más utilizados y eficientes. Fue diseñado específicamente para detectar y corregir errores de un solo bit, y para detectar errores de dos bits. Funciona añadiendo bits adicionales (bits de paridad) al mensaje original en posiciones específicas, determinadas por potencias de dos (1, 2, 4, 8, etc.). Estos bits de paridad se calculan en función de los bits de datos y se utilizan para identificar y corregir cualquier error que pueda ocurrir durante la transmisión.

Es importante mencionar que el algoritmo de Hamming demostró ser bastante robusto y eficaz en la corrección de errores. Observado con los resultados obtenidos, que indican que la eficacia de la implementación del algoritmo es bastante alta, alcanzando en promedio una similitud del 92.41% entre los datos transmitidos y recibidos. Así mismo, la mayoría de las transmisiones lograron una precisión igual o superior al 88%, y que la similitud perfecta fue el resultado más frecuente. Sin embargo, la implementación se podría mejorar, debido a que algunas transmisiones tuvieron una similitud tan baja como el 43%, lo que nos lleva a reflexionar sobre las posibles causas de tales discrepancias y la capacidad inherente del código de Hamming para abordarlas.

El Fletcher Checksum es un algoritmo que genera un código de verificación para detectar errores en bloques de datos. A diferencia de otros métodos de checksum tradicionales que trabajan bit a bit, Fletcher procesa los datos a nivel de byte o palabras más grandes, proporcionando una verificación más rápida y eficiente. Este algoritmo suma consecutivamente los bytes de datos, llevando a cabo un módulo después de cada suma para asegurarse de que el resultado final sea un número de un tamaño determinado. Aunque no está diseñado para corregir errores, el Fletcher Checksum es eficaz en la detección de errores comunes en la transmisión de datos.

Analizando los resultados obtenidos por medio del algoritmo Fletcher Checksum, cabe destacar que se tuvo un total de 10,120 palabras, debido a que se tuvo una muestra repetida 440 veces. El propósito detrás de la repetición de esto fue evaluar la efectividad del algoritmo en condiciones estandarizadas.
Si bien, estadísticamente el modelo se desempeñó correctamente, se tuvo una media de diferencia menor al 20%. En el contexto de redes, la integridad es un aspecto fundamental, así que al tener pérdidas puede tener consecuencias significativas.

Durante la implementación del código que utilizaba el código de Hamming para la corrección de errores, nos encontramos con un resultado inesperado y claramente incorrecto: el archivo similitudes.txt registró únicamente valores 0. Esto significó que no había ninguna similitud entre lo transmitido y lo recibido. En condiciones normales y aún teniendo errores de transmisión, se tuvo que haber conseguido aunque sea una coincidencia. Esto llevó a la revisión de la lógica de la implementación. Una detección de errores tan baja sugería que el problema no era simplemente una casualidad, si no un error en la codificación o codificación de la información. Por lo tanto, se priorizó el proceso de revisión para poder asegurarse que funcionara como se esperaba.

Para futuras réplicas de esta práctica, usando el algoritmo Fletcher Checksum, es importante tomar en consideración que los resultados estadísticos del algoritmo pueden tener una precisión alta, sin embargo puede ser que no sea lo más efectivo en este contexto por las pérdidas. Por lo cual se recomienda explorar técnicas que puedan mejorar la integridad de los datos, por ejemplo combinarlo con otro método de detección de errores para poder reforzar el valor de la precisión. Así mismo, es importante mencionar que durante la implementación, nos encontramos con un desafío técnico relacionado con la manipulación de datos en Python. Específicamente, al intentar unir una lista de datos, el programa arrojó un error que indicaba que estaba tratando de unir un tipo de dato inesperado: bytes en lugar de strings. Esto se debió a la naturaleza binaria de los datos que estábamos tratando y a las particularidades de cómo Python maneja diferentes tipos de datos. La solución a este problema fue decidir y mantener una coherencia en la forma en que se manejaban los datos a lo largo del proceso, ya sea trabajando exclusivamente con strings o bytes. Una vez que se decidió el tipo de datos a utilizar, se ajustó el código para manipular y unir correctamente los datos, lo que resultó en la correcta ejecución del programa.

Para la ejecución en simultáneo del emisor y receptor, se centralizó en un jupyter notebook. Se hizo uso de la biblioteca subprocess ejecutar los scripts al mismo tiempo. De esta forma, se registró claramente y de forma reproducible los pasos a seguir y los procesos realizados. Esto agilizó el flujo de trabajo, al unificar la ejecución y el análisis en un solo entorno, así como también ayudó a la detección temprana y corrección de errores, como los que mencionaron con anterioridad.

Comentario grupal sobre el tema:

El desarrollo de este laboratorio fue de mucho aprendizaje para el equipo, conformado por Alejandro "Ale" Gómez y Juan Carlos "Juanca" Baján. Ale trabajó e investigó del algoritmo de Hamming, ajustándose a la perfección para nuestras necesidades, mientras que Juanca implementó el algoritmo de Fletcher Checksum. Simular estas pruebas ayudó a entender cómo se desplegaría todo en situaciones prácticas. Como se discutió en clase, se optó por hacer solo 1,000 pruebas en lugar de las inicialmente planteadas 10,000, porque, siendo realistas, tomaría demasiado tiempo. Usar Jupyter Notebook fue una decisión acertada, ya que hizo que el análisis estadístico fuera más efectivo, además de poder dejar los gráficos incrustados en el código. Con herramientas como Seaborn, Numpy y Pandas, el análisis realizado fue de mucho aprendizaje y ayudó a sacar conclusiones. Después de todo el proceso, algo genial sucedió: Ale y Juanca se sentaron a repasar el código del otro, asegurándose de que ambos tuvieran una comprensión completa de estos esquemas de detección y corrección de errores.

Conclusiones:

- La práctica revela la importancia crítica de tener esquemas de detección y corrección de errores robustos, especialmente en canales propensos a interferencias o ruido. Si bien el código de Hamming se mostró efectivo en la corrección de errores de un solo bit y la detección de errores de dos bits, algoritmos como Fletcher Checksum demostraron ser rápidos y eficientes en la detección de errores, pero sin la capacidad de corrección.

- Al simular un canal no confiable y aplicar ruido a la trama, se obtuvo un acercamiento sobre cómo se podrían llegar a comportar estos algoritmos dentro de un escenario real. La automatización de las pruebas y el análisis estadístico ayudaron a comprender por qué es importante realizar distintas pruebas muy minuciosas en distintos escenarios y condiciones para poder medir la eficacia que tienen los algoritmos probados.

- Al abordar problemas relacionados con la comunicación de datos, especialmente cuando se trabaja en bajo nivel, es esencial tener claridad sobre los tipos de datos con los que se está trabajando en cada paso del proceso. Es recomendable realizar pruebas y verificaciones constantes para asegurar que los datos se estén manipulando y transmitiendo correctamente.

- Cabe destacar que mientras que el algoritmo Fletcher Checksum muestra una precisión mediana del 90%, con un cuarto de sus transmisiones por debajo del 84%, el algoritmo de

Hamming presentó una precisión mediana del 94%. Aunque ambos son efectivos, en términos de precisión y calidad de transmisión, el algoritmo de Hamming demostró tener una mejor implementación frente a Fletcher Checksum en contextos críticos.

Citas y Referencias:

Tanenbaum, A. S. (2010). Redes de computadoras (5ta edición). Pearson.

Forouzan, B. A. (2007). Data communications and networking (4ta edición). McGraw-Hill.

Stallings, W. (2013). Data and computer communications (10ta edición). Pearson.