Informe de experimentación Dual CheckSum y CRC

Nombres:

Oscar Stiven Muñoz Ramirez
Diego Armando Polanco Lozano
Ricardo Andres Chamorro
Luis Manuel Rojas Correa

Computación y Estructuras Discretas II
Uram Anibal Soza **|** Angela Patricia Villota

Cali, Universidad ICESI
31/05/2024

Índice

1. Preámbulo

En las ciencias computacionales y las telecomunicaciones, es imprescindible validar la fidelidad de los datos transferidos a través de un canal de comunicación catalogado como no confiable, como lo son las redes de internet. Razón por la cual, existen algoritmos para generar códigos de verificación de errores en los datos transmitidos, entre los que se encuentra Dual-CheckSum, y Cyclic Redundancy Check.

Para lograr el objetivo principal de esta tarea integradora, -desarrollar y analizar el comportamiento de los algoritmos mencionados en un ambiente de pruebas controlado- se han definido 4 conjunto de escenarios que representan las distintas entradas -mensajes en Bits para calcular su checksum- que se diferencian en tamaño:

- Toy, $n < 10^2$
- Pequeñas, $10^2 \leq n < 10^4$
- Medianas, $10^4 \leq n < 10^5$
- Grandes, $n \geq 10^6$

Conociendo que la complejidad temporal teórica tanto del Dual-CheckSum como del Cyclic Redundancy Check es O(n) (lineal), y con el ánimo de comprender el efecto que tiene el tamaño de las particiones en bloques en el tiempo de ejecución del algoritmo, se estipulo para ambos algoritmos diferentes los siguientes valores de K [1]: 8,16,32,64. Además de calcularse el módulo con la fórmula de fletcher: m: $2^k - 1$

Es importante conocer que para el tamaño Toy y Small se hace uso de dos gráficos en unidades de tiempo diferentes al ser pruebas de bytes muy pequeños, en adelante se prescinde de la unidad nanosegundo.

[1]Cantidad de bits representadas en el módulo, y cantidad de particiones (data blocks)

2. Objetivos de las Pruebas

1. Verificar la correcta implementación de los algoritmos de Checksum y CRC.
2. Medir el tiempo de ejecución de los algoritmos con diferentes tamaños de entrada.
3. Comparar la complejidad experimental con la complejidad teórica de los algoritmos.
4. Determinar la relación de los distintos valores de K y su respectivo Módulo en el Tiempo de Ejecución Experimental del Algoritmo

3. Alcance del Modelo Experimental

Este informe sube únicamente las funcionalidades del Dual-CheckSum, y Cyclic Redundancy Check definidos, y probados, dentro del marco de la Tarea Integradora II del curso Computación y Estructuras Discretas; el enunciado. CyEDII_2024-1_TI2.docx

3.1 Entornos de Experimentación

Los algoritmos fueron ejecutados en la computadora con las siguientes caracterìsticas, dentro del IDE Intellij Idea:

PC	Hp Pavilion
Procesador	11 th Gen Intel(R) Core(™) i5 -1135G7 @ 2.40 GHz
RAM	8 GB
OS	Windows
Pruebas encargadas	Toy test, Small test, Medium Test, Big test

Durante la experimentación el equipo no tuvo subprocesos abiertos, ni fue utilizado para cualquier otra actividad diferente a la de compilar ambos algoritmos.

4. Metodología de la Experimentaciòn

Cada algoritmo fue probado de forma independiente para cada uno de los datasets generados, es decir que ambos algoritmos nunca compilaron al mismo tiempo, y analizaron cada uno de los archivos por separado. El tiempo de ejecuciòn se determinó únicamente para los métodos desarrollados que implementan los algoritmos, es decir que no se calculo el tiempo que demoraba la generaciòn de los datasets, la conversión de tipo de dato de String a Byte, o el tiempo necesitado para las particiones en bytes de los mensajes; además de los datos considerados atípicos. Tomando en cuenta el enfoque y metodología utilizada por Philip Koopman: Koopman, P. (2024). *An Improved Modular Addition Checksum Algorithm*. arXiv. https://doi.org/10.48550/arXiv.2304.13496

5. Análisis de los Resultados Experimentales

Como se mencionó anteriormente, el objetivo supone analizar en detalle la relación entre el tamaño de la entrada de datos, el tiempo de ejecución de los algoritmos de Checksum y CRC, y las particiones K de los datos definidos. Para el caso del dual checksum, el módulo M seleccionado será calculado mediante el enfoque de Fletcher m:$2k - 12^{k} - 12k - 1$. Tomando en cuenta la eficiencia y la precisión, se seleccionaron los valores de k= 8, 16, 32, 64k = 8, 16, 32, 64k = 8, 16, 32, 64 debido a que representan tamaños comunes en la arquitectura de los sistemas de computación, permitiendo una evaluación coherente y comparativa del rendimiento de los algoritmos en diferentes escalas de datos (Koopman, 2024). Por lo tanto, se plantean las siguientes preguntas y consideraciones en las observaciones:

- Se espera observar cómo aumenta el tiempo de ejecución a medida que aumenta el tamaño de la entrada. Es importante identificar si hay alguna regularidad o tendencia clara.
- A través del análisis gráfico y matemático, se determinará la naturaleza de la relación. Esto puede incluir análisis de regresión para identificar si la relación es lineal, cuadrática, exponencial, etc.
- Se aplicarán diferentes modelos de ajuste de curvas para encontrar la función que mejor describa los datos experimentales.
- Se comparará la función de ajuste obtenida de los datos experimentales con la complejidad teórica esperada de los algoritmos (O(n), O(n^2), etc.).
- Se evaluará si la complejidad teórica del algoritmo coincide con los resultados observados en las pruebas de rendimiento.

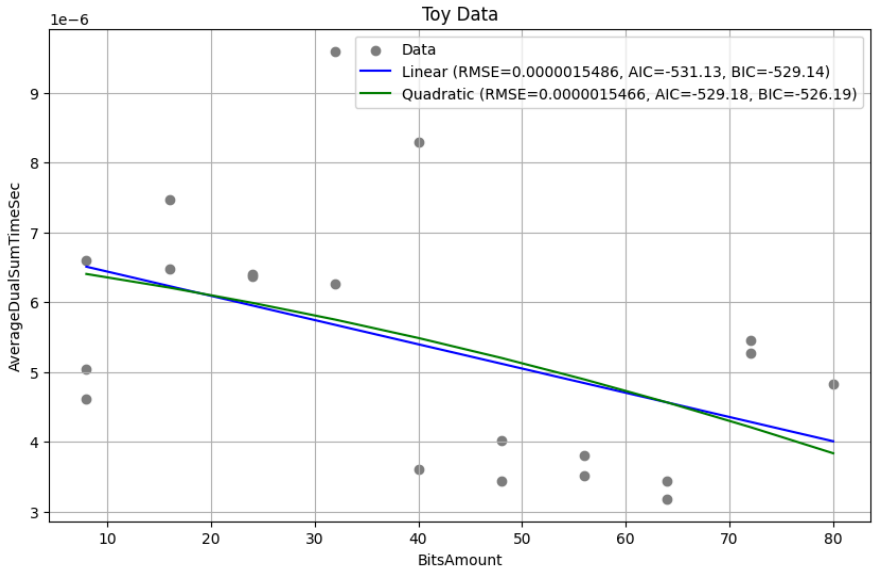
- Se analizarán los datos para identificar cualquier tamaño de entrada específico que cause un comportamiento anómalo en el tiempo de ejecución.
- Se considerarán factores como las condiciones del entorno de pruebas, variaciones en el hardware, y cualquier otra posible fuente de error que pudiera influir en los resultados.

Este análisis detallado permitirá entender mejor el rendimiento y las limitaciones de los algoritmos de Checksum y CRC, y proporcionar recomendaciones para su uso en diferentes contextos de aplicación.

5.1 Comportamiento Gráfico de los algoritmos.

5.1.2 Comportamiento del Dual CheckSum

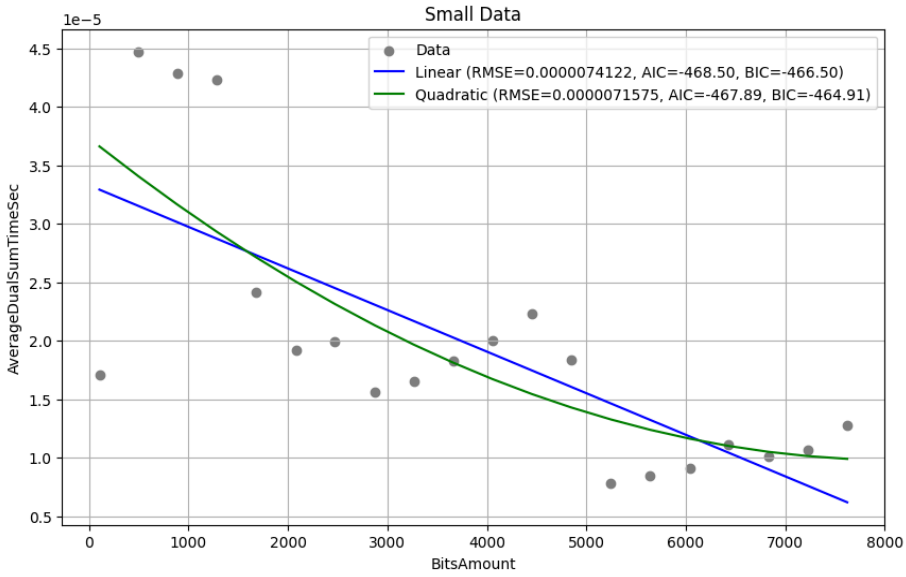
5.1.2.1.1 Dual Check Sum Toy Data con K=8



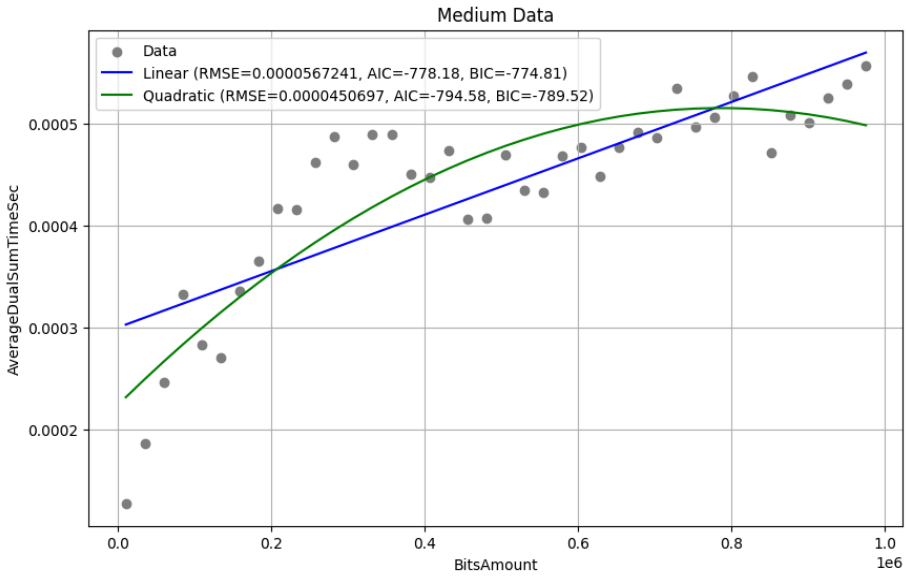
Gráfica 1: Toy Data Dual CheckSum con K =8

Elaboraciòn Propia

5.1.2.1.2 Dual Check Sum Small Data con K=8



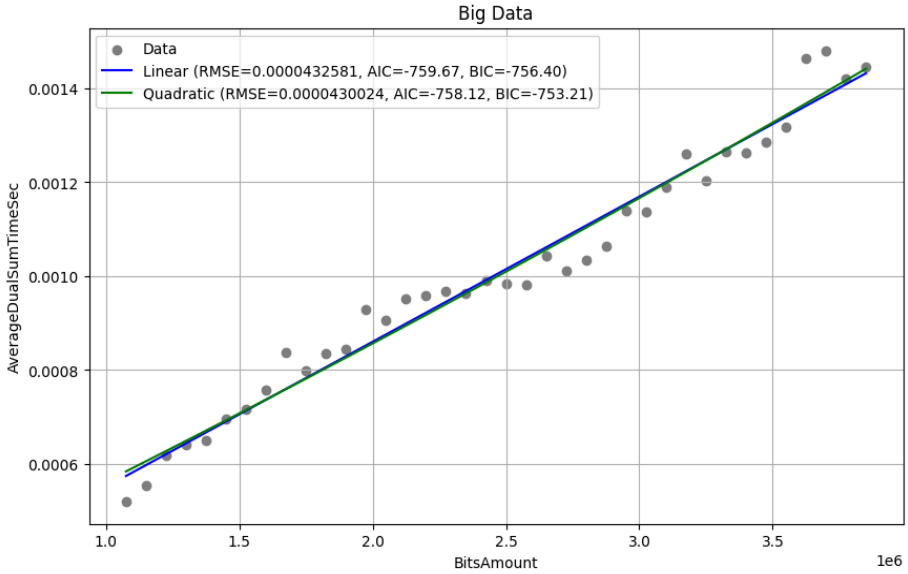
5.1.2.1.3 Dual Check Medium Data con K=8



Gráfica 2: Medium Data Dual CheckSum con K =8

Elaboraciòn Propia

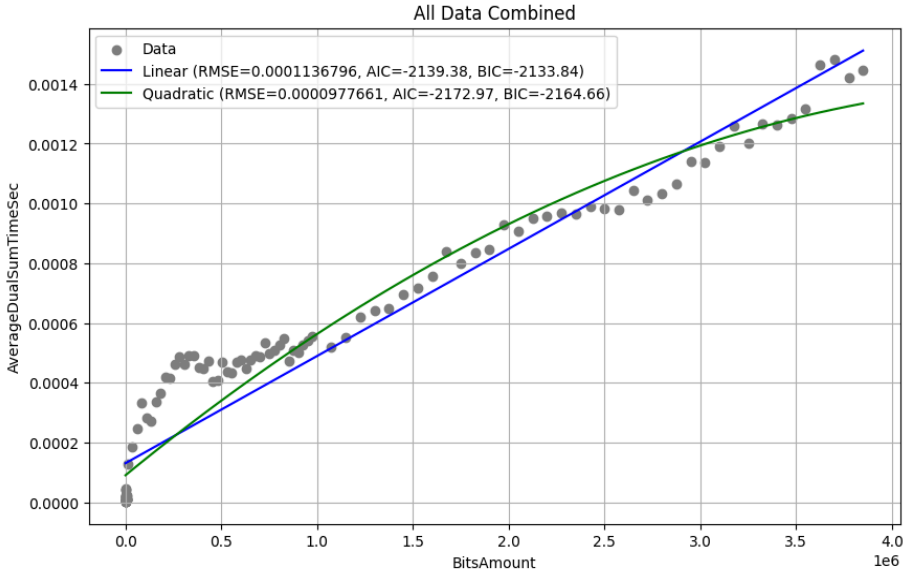
5.1.2.1.4 Dual Check Big Data con K=8



Gráfica 3: Big Data Dual CheckSum con K =8

Elaboraciòn Propia

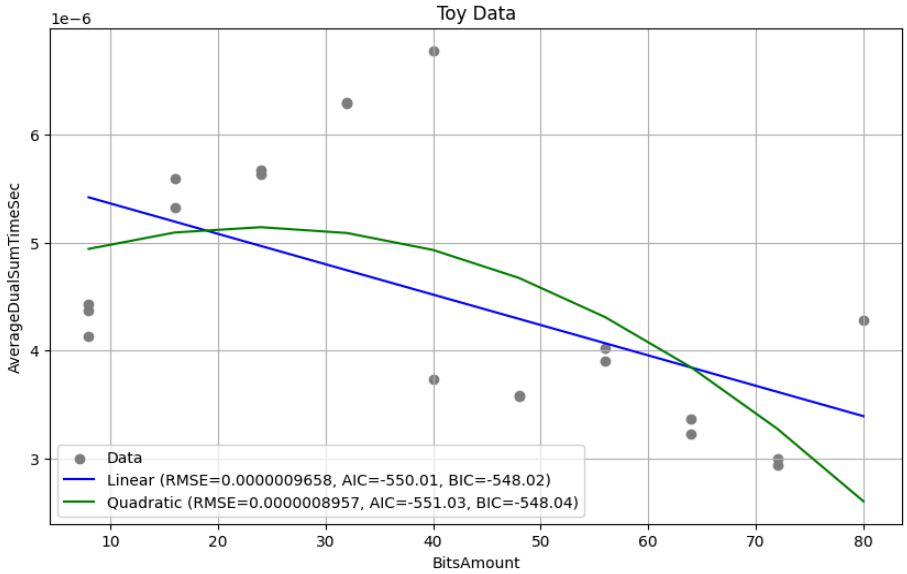
5.1.2.1.5 Dual Check Data Combined con K=8



Gráfica 4: All Data Combined Dual CheckSum con K =8

Elaboraciòn Propia

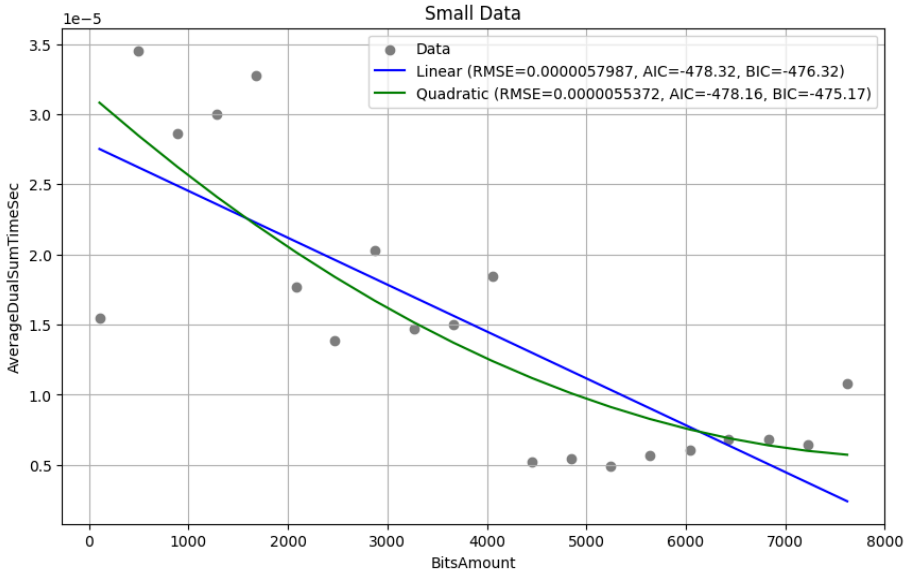
5.1.2.2.1 Dual Check TOY Data con K=16



Gráfica 5: Big Data Dual CheckSum con K =16

Elaboraciòn Propia

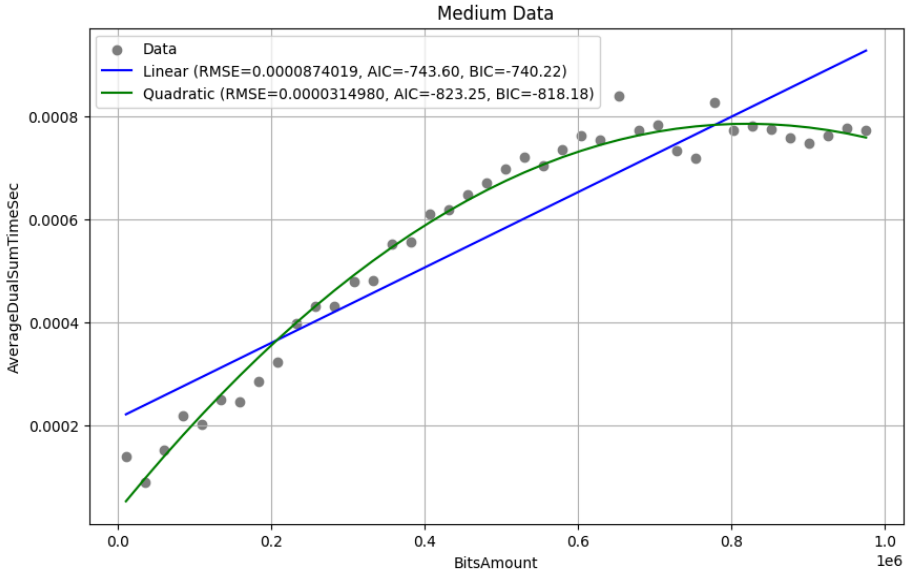
5.1.2.2.2 Dual Check Small Data con K=16



Gráfica 6: Small Data Dual CheckSum con K =16

Elaboraciòn Propia

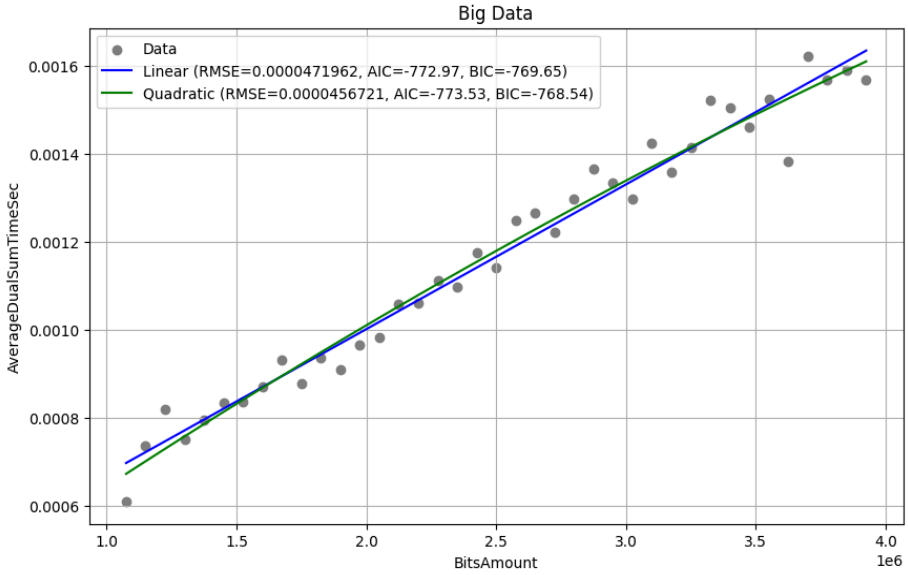
5.1.2.2.3 Dual Check Medium Data con K=16



Gráfica 7: Medium Data Dual CheckSum con K =16

Elaboraciòn Propia

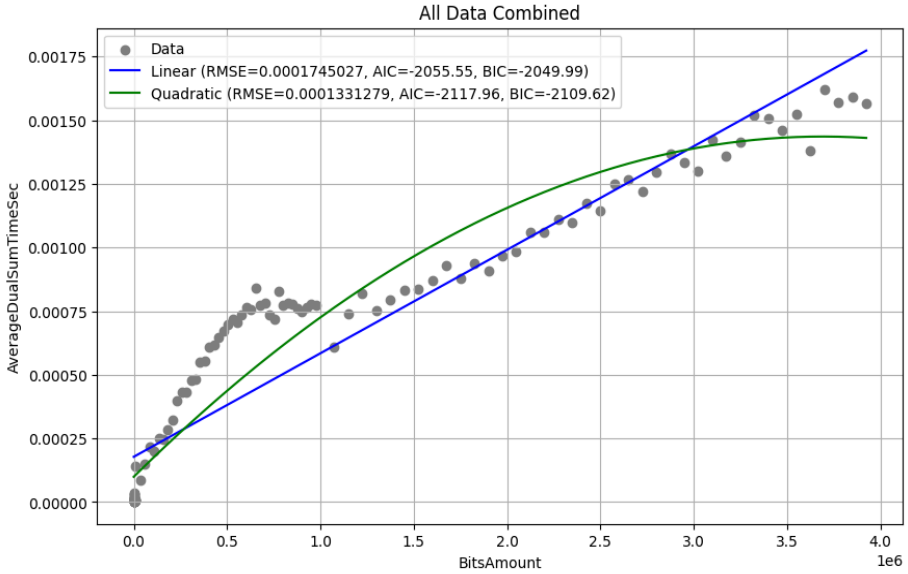
5.1.2.2.4 Dual Check Big Data con K=16



Gráfica 8: Big Data Dual CheckSum con K =16

Elaboraciòn Propia

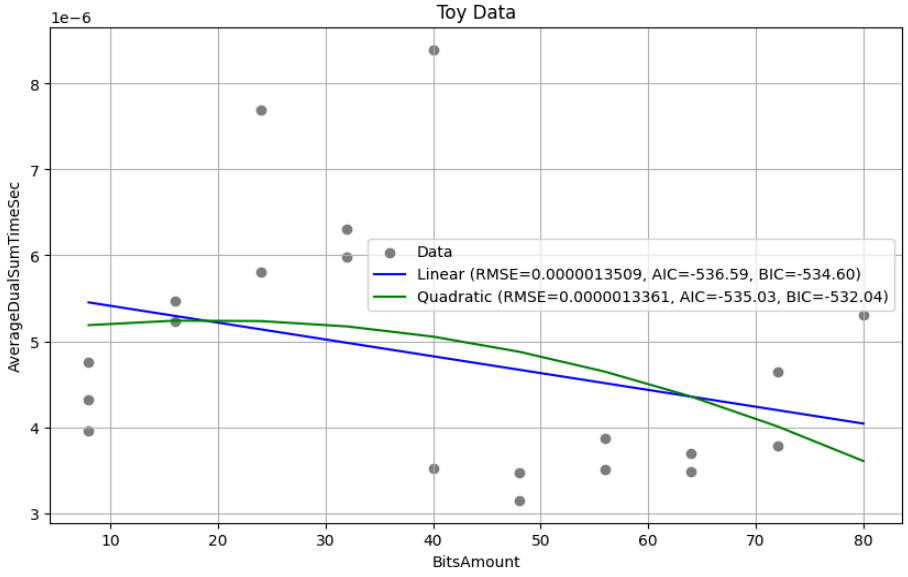
5.1.2.2.5 Dual Check All Data combined Data con K=16



Gráfica 9: All Data Combined Dual CheckSum con K =16

Elaboraciòn Propia

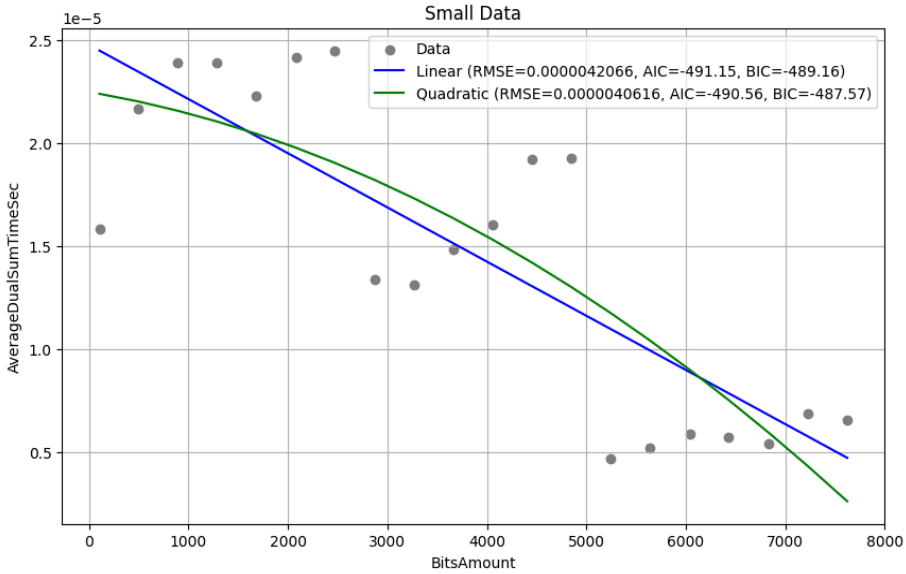
5.1.2.3.1 Dual Check Toy Data con K=32



Gráfica 10: Toy Data Dual CheckSum con K =32

Elaboraciòn Propia

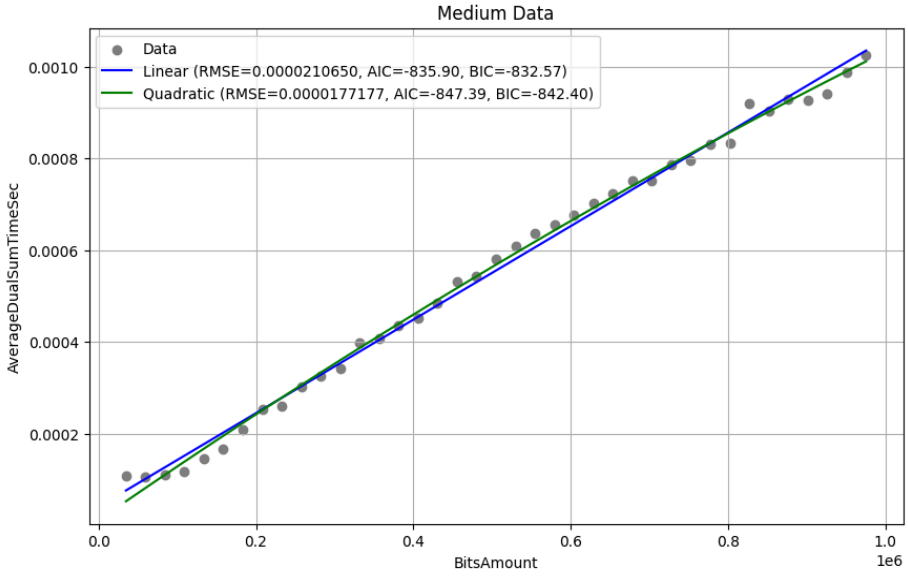
5.1.2.3.2 Dual Check Small Data con K=32



Gráfica 11: Small Data Dual CheckSum con K =32

Elaboraciòn Propia

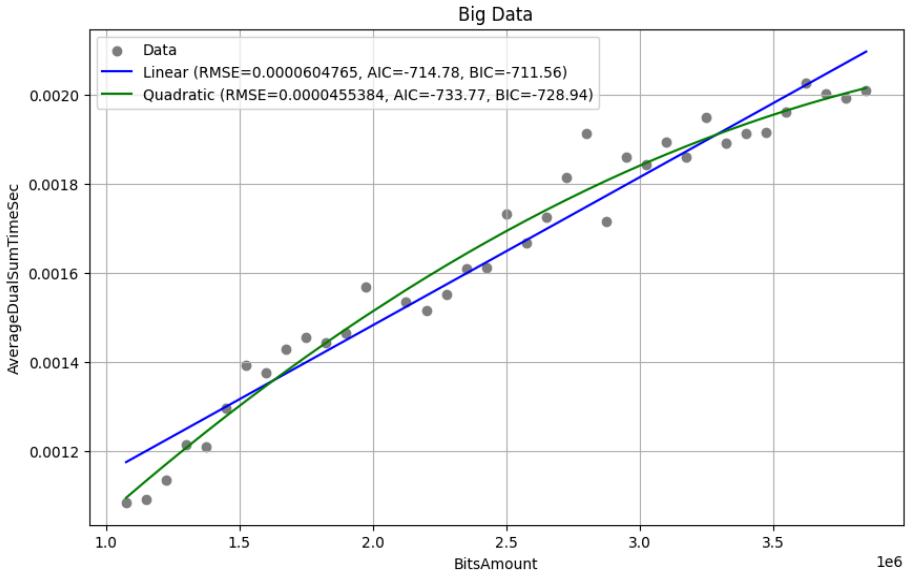
5.1.2.3.3 Dual Check Medium Data con K=32



Gráfica 12: Medium Data Dual CheckSum con K =32

Elaboraciòn Propia

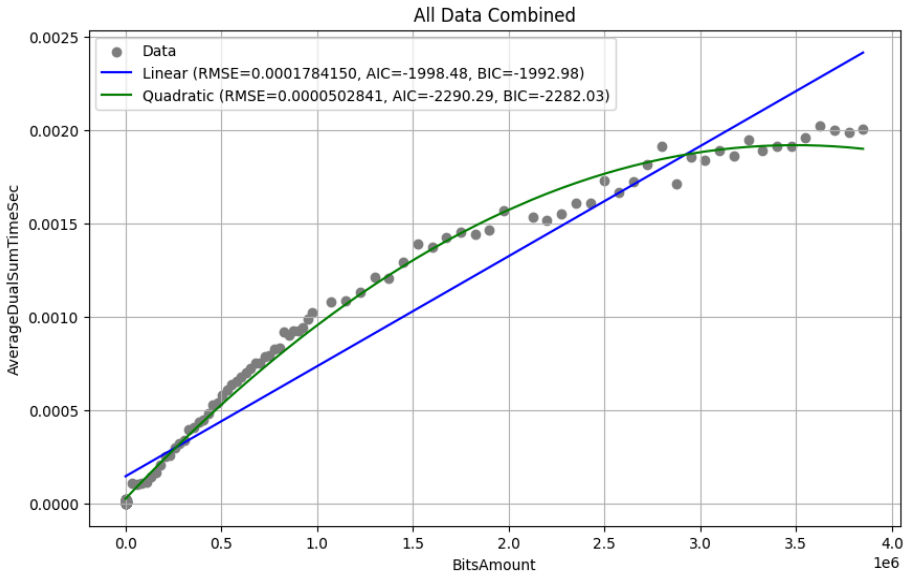
5.1.2.3.4 Dual Check Big Data con K=32



Gráfica 13::Big Data Dual CheckSum con K =32

Elaboraciòn Propia

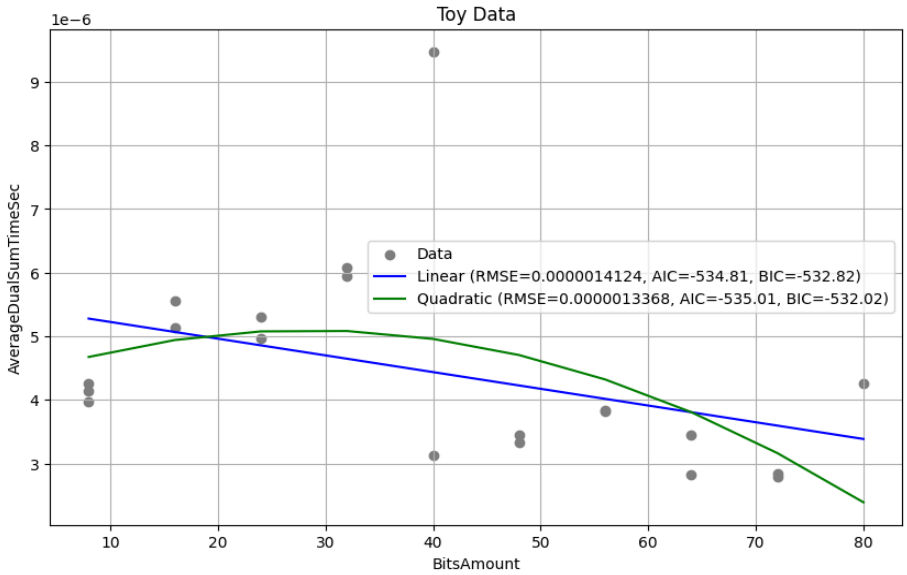
5.1.2.3.5 Dual Check All Data combined con K=32



Gráfica 14:All Data Combined Dual CheckSum con K =32

Elaboraciòn Propia

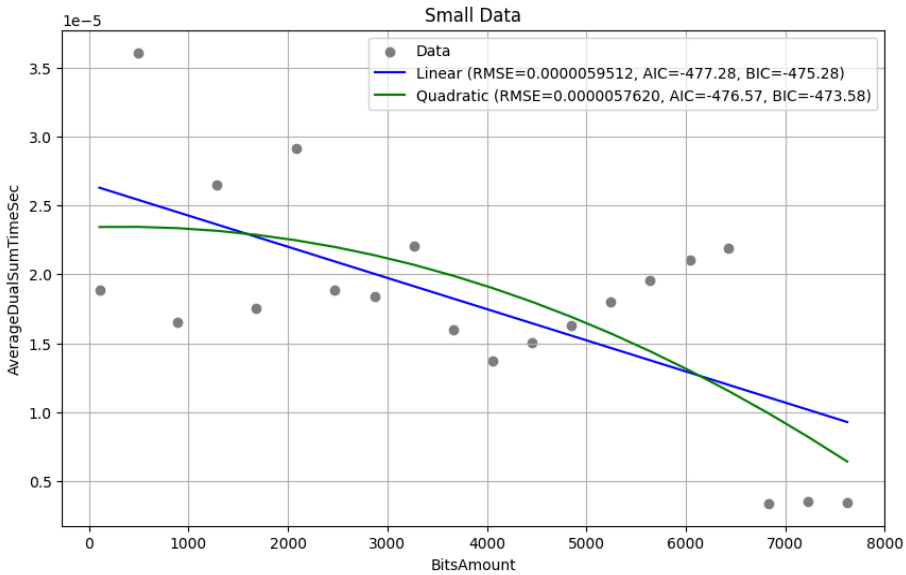
5.1.2.4.1 Dual Check Big Data con K=64



Gráfica 15:Toy Data Combined Dual CheckSum con K =64

Elaboraciòn Propia

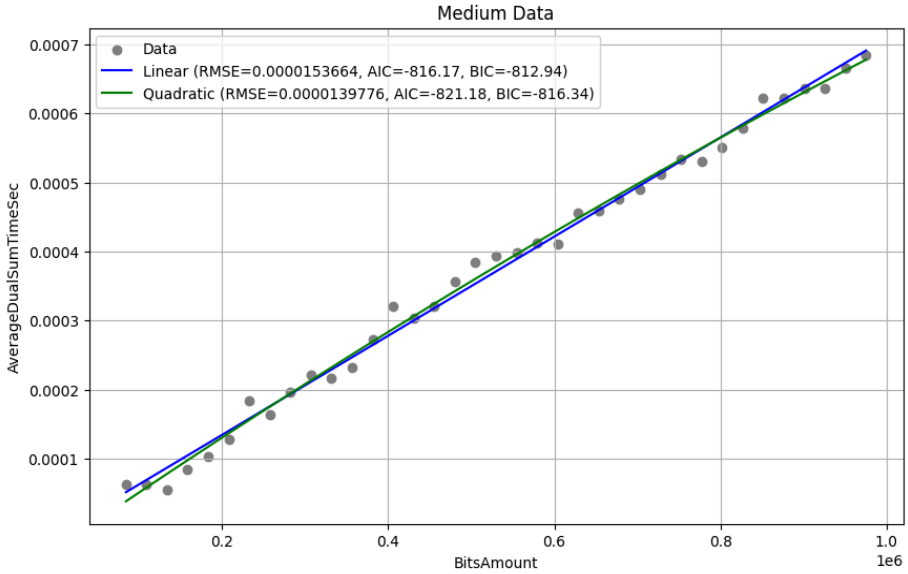
5.1.2.4.2 Dual Check Big Data con K=64



Gráfica 16:Small Data Combined Dual CheckSum con K =64

Elaboraciòn Propia

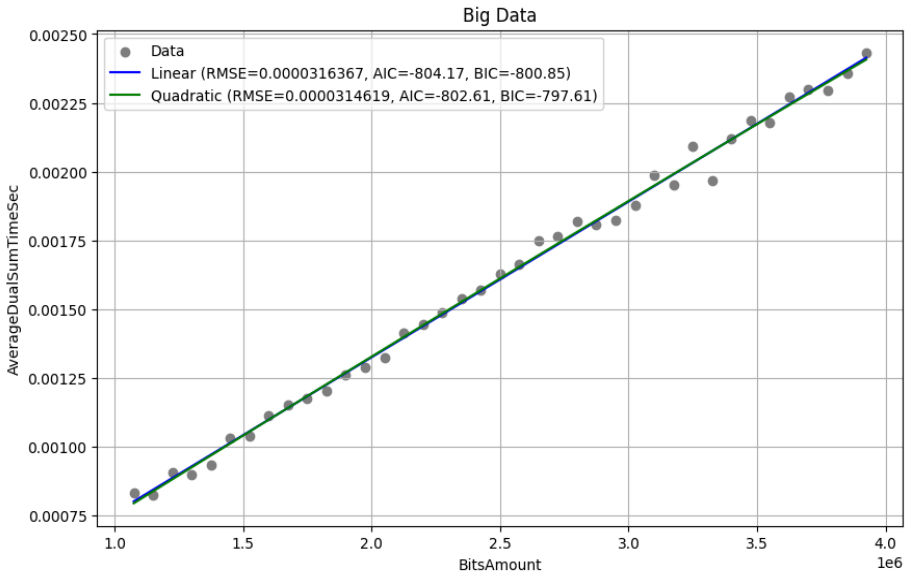
5.1.2.4.3 Dual Check Medium Data con K=64



Gráfica 17:Medium Data Combined Dual CheckSum con K =64

Elaboraciòn Propia

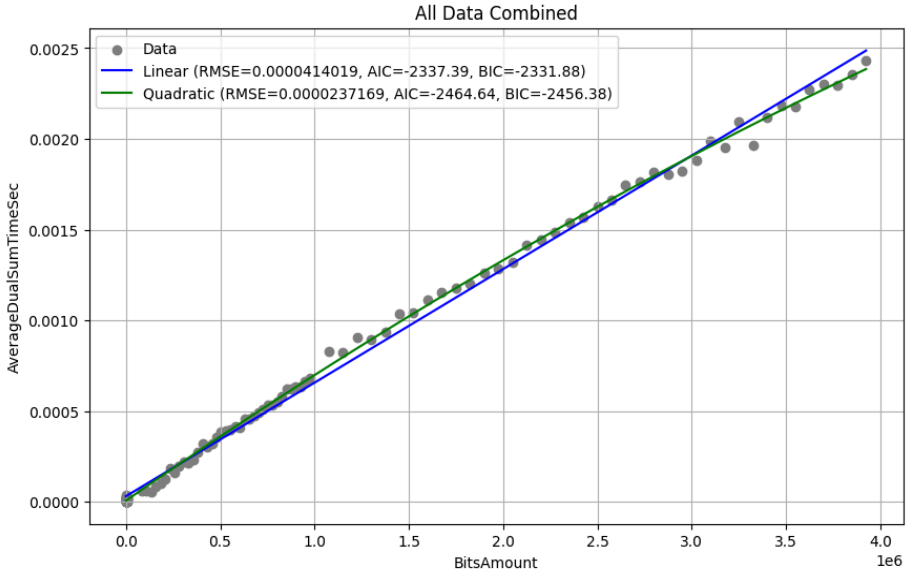
5.1.2.4.4 Dual Check Big Data con K=64



Gráfica 18:Big Data Dual CheckSum con K =64

Elaboraciòn Propia

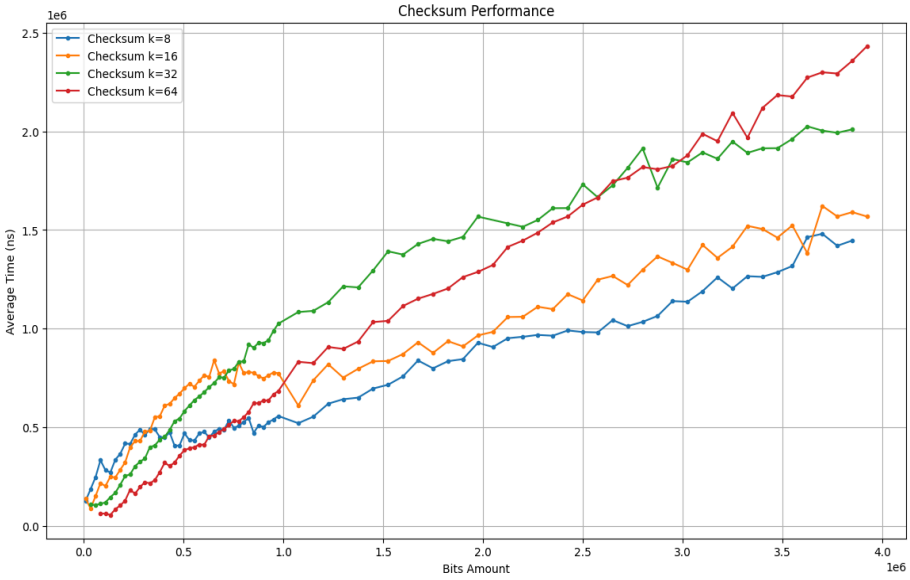
5.1.2.4.5 Dual Check Big Data con K=64



Gráfica 19:All Data Combined Dual CheckSum con K =64

Elaboraciòn Propia

5.1.2.5 Comparativa de todos los Dual Check Sum



Gráfica 20:All Data Combined of every Dual CheckSum

Elaboraciòn Propia

5.1.3 Análisis del Comportamiento Dual CheckSum

Si bien el comportamiento teórico del dual checksum es lineal $O(n)$, el tiempo encontrado en la ejecución del dual checksum presentaba variaciones significativas dependiendo del valor k elegido, y por consiguiente su módulo asignado utilizando la fórmula de Fletcher.

Los resultados de rendimiento temporal del Dual CheckSum, evidenció una correlación inversa entre el tiempo que la función demoraba ejecutar una cantidad de datos para un k escogido, es decir que a medida que se incrementa el tamaño de K, el tiempo de ejecución disminuye. Una causante de esta relación inversa, podría suponer que dividir el mensaje en una mayor cantidad de bloques, disminuye la cantidad de datos (bits) a la que la función deberá calcular su dual checksum, permitiendo reducir el tiempo necesario para calcular el checksum de cada bloque individual y sumarlos.

Esta relación se evidencia en los gráficos del comportamiento del Dual CheckSum para diferentes valores de K (8, 16, 32, 64). En todos los casos, se muestra una tendencia a la reducción del tiempo de ejecución a medida que se incrementa k.

Valor K	Cantidad y Proporción de Bits a analizar
K: 8	$Cantidad\ de\ Bits\ en\ cada\ Block = \frac{125000\ Bytes}{8} = 15\,625$ $Proporción\ del\ total\ (10^6\ bits) = \frac{15\,625 * 100}{125\,000} = 12.5\ \%$
K: 16	$Cantidad\ de\ Bits\ en\ cada\ Block = \frac{125000\ Bytes}{16} = 7\,812.5 \simeq 7813$ $Proporción\ del\ total(10^6\ bits) = \frac{7\,813 * 100}{125\,000} = 6.25\ \%$
K: 32	$Cantidad\ de\ Bits\ en\ cada\ Block = \frac{125\,000\ Bytes}{32} = 3\,906.25 \simeq 3907$ $Proporción\ del\ total(10^6\ bits) = \frac{3907 * 100}{125\,000} = 3.1256\mathbf{\%}$
K: 64	$Cantidad\ de\ Bits\ en\ cada\ Block = \frac{125\,000\ Bytes}{64} = 1953.125 \simeq 1954$ $Proporción\ del\ total(10^6\ bits) = \frac{1954 * 100}{125\,000} = 1.56\%$

Al realiza un análisis de la cantidad y proporción de los de Bytes y Bits que la función debe procesar para un caso hipotético de 10^6 bits (125,000 bytes), se evidencia que para los anteriores valores de k (8,16,32,64) a medida que incrementa el valor de k, el dual checksum tendrá que ejecutar una menor cantidad de Bits, lo cual, pese haber más particiones, disminuye el tiempo total de ejecución.

Así mismo, en el artículo de Philip Koopman mencionado en el documento (Koopman, 2024) trata sobre la mejora en los algoritmos de checksum mediante la adición modular, y sugiere que un aumento en el tamaño de K no necesariamente conduce a una pérdida de información. De hecho, procesar bloques más grandes (k bits) puede capturar una variedad de errores comunes, incluyendo errores de un solo bit y errores de ráfaga, mejorando así la detección de errores sin comprometer la integridad de la información transmitida, por lo que el uso de un K mayor reduce la probabilidad de encontrarse con este tipo de anomalías.

Análisis Entre Checksums:

El gráfico muestra el tiempo promedio de ejecución en (NS) del cálculo del checksum para distintos valores de k (8,16,32,64) en función de la cantidad de bits procesados. A continuación se presentan algunas observaciones clave:

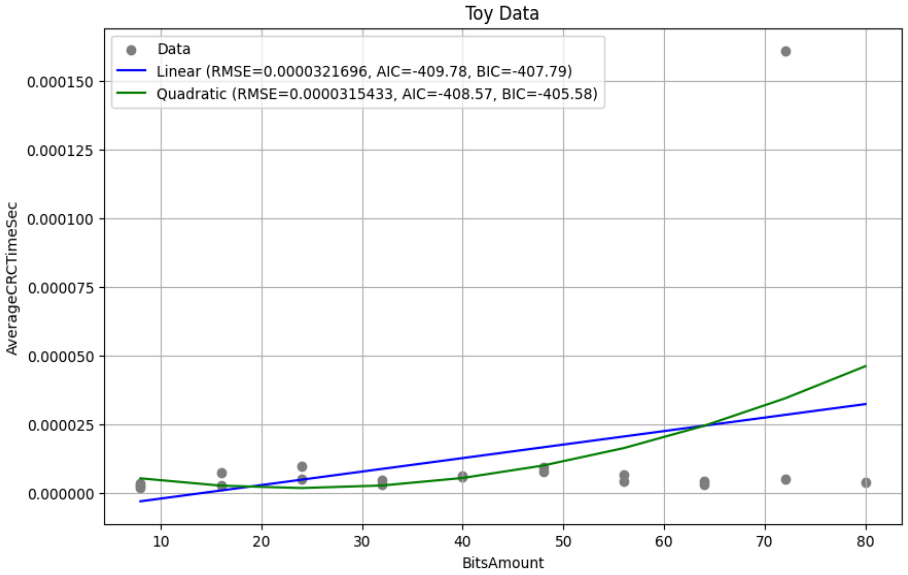
1. **Tendencia general:**
    - A medida que aumenta la cantidad de bits procesados, el tiempo de ejecución promedio aumenta para los valores de k. Esto es esperado, dado que procesar más datos requiere más tiempo.
2. **Comparación entre Valores de K:**
    - Para los valores de K pequeños: Se observa que los checksums con k=8 y=16 tienen un rendimiento mejor (menor tiempo de ejecución) en comparación con los valores más grandes de K cuando los valores de entrada de datos son tipo Big.
    - Para valores de k grandes: Los checksums con k=32 y=64 presentan tiempos de ejecución mayores en comparación con valores menores de k, no obstante cuando son de tipo Medium los datos son más veloces.

Los checksum con valores de k pequeños son más eficientes en términos de tiempo cuando los datos son muy grandes. Debido a que con valores más pequeños de k, hay más bloques pequeños que se pueden procesar rápidamente, ya que los k mas grandes tinder a aumentar significativamente el tiempo de procesamiento debido a la mayor complejidad de manejar bloques más grandes.

Una hipótesis para explicar este fenómeno es que, con tamaños de k grandes, hay menos bloques que analizar cuando los datos son medianos. Esto resulta en una menor exigencia computacional en este contexto. En cambio, cuando los datos son grandes, un mayor k implica que cada operación de procesamiento abarca más bits, lo que aumenta el tiempo total de ejecución debido a la mayor cantidad de datos a analizar.

5.1.4 Comportamiento del CRC

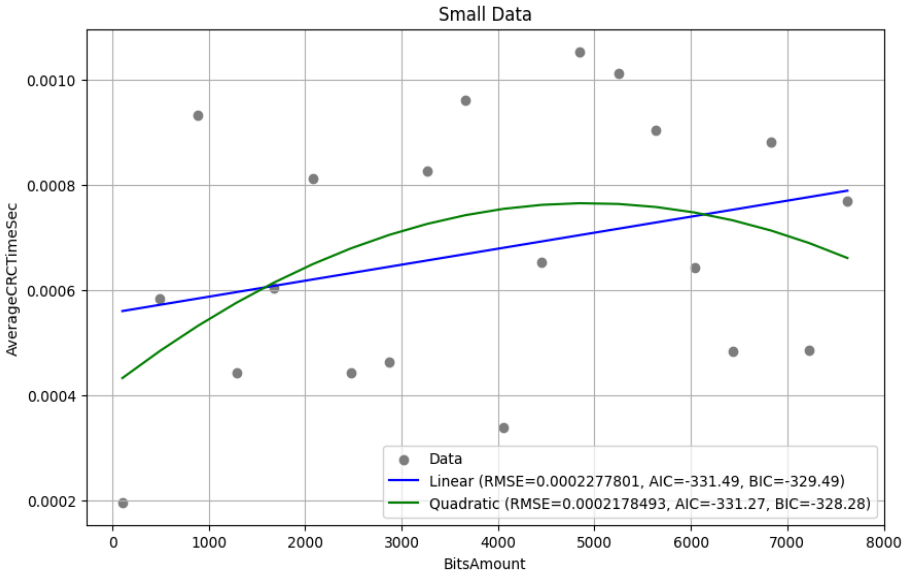
5.1.4.1 CRC - Data toy



Gráfica 21:Toy Data CRC

Elaboraciòn Propia

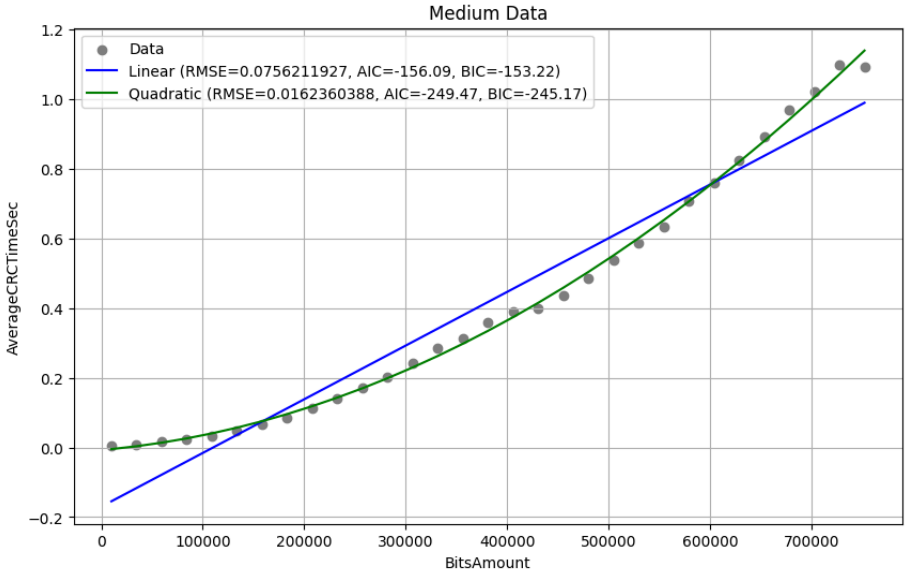
5.1.4.2 CRC - Data Small



Gráfica 22:Small Data CRC

Elaboraciòn Propia

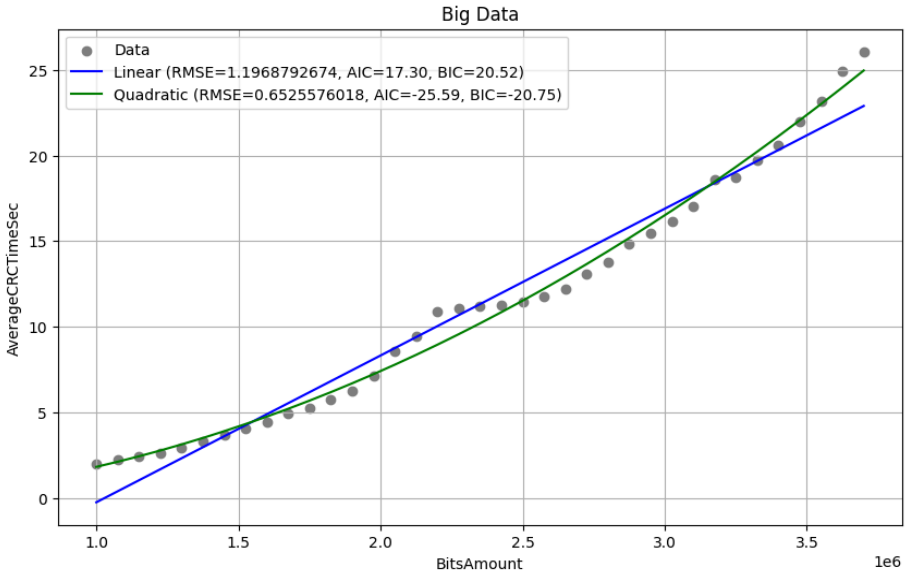
5.1.4.3 CRC - Data Medium



Gráfica 23:Medium Data CRC

Elaboraciòn Propia

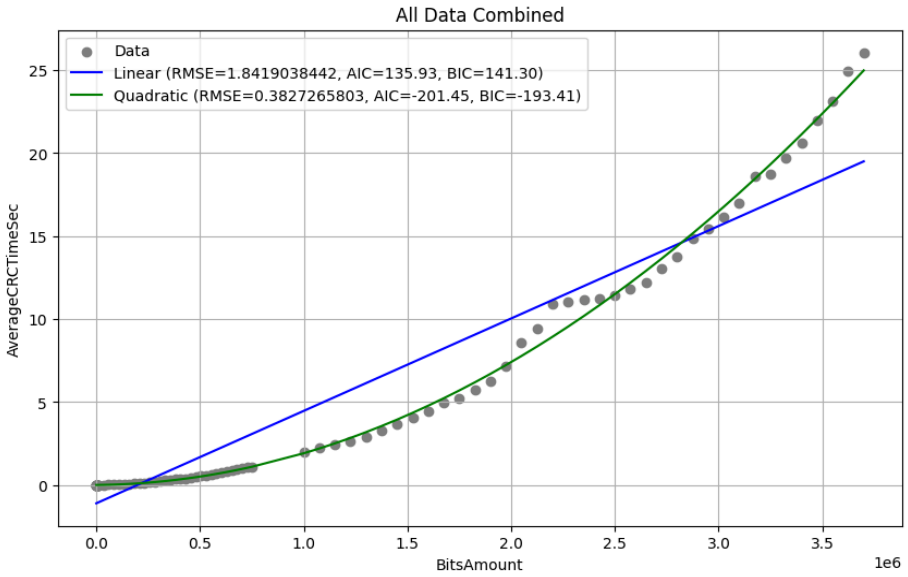
5.1.4.4 CRC - Data Big



Gráfica 23:Big Data CRC

Elaboraciòn Propia

5.1.4.5 CRC - Data Combined



Gráfica 24:All data combined CRC
Elaboraciòn Propia

Comportamiento esperado de CRC

El algoritmo CRC implementado muestra un comportamiento, en donde existe un ajuste cuadrático aproximado, debido al enfoque y tipos de datos en el que se calcula el residuo, así como las operaciones involucradas del lenguaje. Los puntos principales en donde se sustenta lo anterior son:

Operaciones con Tipo de Dato Strings:

El CRC se calcula utilizando operaciones sobre cadenas de caracteres simulando bits (String). En cada paso, se realizan operaciones de XOR bit a bit entre cadenas de la misma longitud que el polinomio divisor. Estas operaciones no solo son costosas en términos de tiempo de procesamiento, sino que también aumentan en complejidad a medida que aumenta el tamaño de la cadena de entrada.

Manipular cadenas de caracteres en representación de los bits, incremente la complejidad del algoritmo, en especial para las operaciones XOR entre cadenas de caracteres, ya que, además de usar un tipo de dato distinto a bit, el número de iteraciones corresponderá a la longitud del mensaje original y del polinomio divisor, correspondiendose que para Strings de una mayor longitud se requiera una mayor cantidad de procesamiento.(Peterson y Brown, 1961)

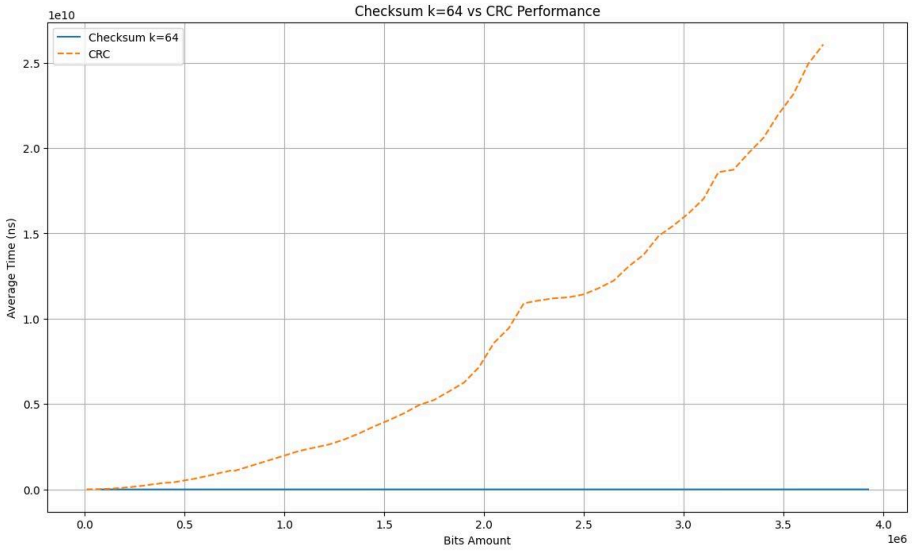
Tomando en cuenta que la función XOR mapea y compara cada uno de los bits para la cadena String de entrada, empleando la función del lenguaje scala zio, se deduce que cada par de bits son comparados para la creación de una nueva cadena de resultados. La operación anterior tiene

teóricamente un comportamiento lineal O(n). No obstante, en los casos donde estas operaciones se repiten iterativamente para cada segmento del mensaje, la complejidad temporal puede llegar a ser significativa. (Mattamorphic, 2023). Es decir, que la continua comparación y construcción de nuevas cadenas, incrementa el costo computacional, reflejándose en el tiempo total empleados en manejar mensajes largos en recursos limitados.

Ajuste del Resto

Los ajustes continuos a las cadenas de restos, las iteraciones implican operaciones de concatenación y sustracción en la cadena de restos. Si bien las operaciones tienen una complejidad temporal lineal O(n) con respecto al tamaño del mensaje, el número total de operaciones puede crecer cuadráticamente (Mattamorphic, 2023). La "acumulación" de estas operaciones puede llevar a un crecimiento cuadrático. Este comportamiento cuadrático es una característica inherente del enfoque iterativo utilizado en el cálculo del CRC, y destaca la necesidad de explorar técnicas de optimización para minimizar el impacto en el rendimiento (Peterson y Brown, 1961).

6. Comparación Dual CheckSum y CRC



Gráfica 25: Comparación entre Dual CheckSum vs CRC
Elaboraciòn Propia

Como puede observarse en la gráfica anterior, el algoritmo Dual CheckSum implementado, para un valor k de 64, posee un menor tiempo de ejecución que el Cyclic Redundancy Check, y se aproxima mejor a su tiempo teórico de complejidad temporal lineal O(n), para cuando

incrementan el tamaños de los Bits que deben ser procesados. Puede decirse que este fenómeno radica en el tipo de dato seleccionado en la implementación de ambos algoritmos, ya que como se vió anteriormente el hecho de que el Cyclic Redundancy Check fuese implementado con Strings -en representación de los bits- supone un incremento semejante al cuadrático debido a las razones explicadas en el apartado "Comportamiento Esperado del CRC". De otra parte, si bien el dual CheckSum posee un comportamiento experimental que tiende a la linealidad, este dependerá del K -cantidad de bits representadas en el módulo, y por ende, cantidad de data blocks generados- ya que, de este incremento, se logrará un tiempo menor de ejecución y aproximado al teórico lineal, ya que se reducirán significativamente la cantidad de bits que deberá analizar en cada iteración de la sumatoria; este aumento en el tamaño de K no necesariamente conduce a una pérdida de información, ya que, como se argumentó anteriormente, y de acuerdo al artículo de Koopman citado, procesar bloques más grandes (k bits) puede capturar una variedad de errores comunes, incluyendo errores de un solo bit y errores de ráfaga.

Recomendaciones

1.Optimización **de algoritmos:**

Dual Checksum:

- Continuar optimizando el algoritmo dual checksum para mejorar su rendimiento en tamaños de bloques mayores. Y evaluar la posibilidad de ajustar con diferentes valores de K en función del tamaño de entrada se puede lograr un equilibrio óptimo entre el tiempo de ejecución y la precisión de la detección de errores.

CRC:

- Investigar alternativas a la representación de bits mediante strings para el CRC. El uso de tipos de datos nativos (e.g., List de bits) podría reducir significativamente el tiempo de ejecución y la complejidad computacional.

Referencias

Koopman, P. (2024). *An Improved Modular Addition Checksum Algorithm*. arXiv. https://doi.org/10.48550/arXiv.2304.13496

Peterson, W.W., & Brown, D.T. (1961). Cyclic Codes for Error Detection. Proceedings of the IRE, 228-235. https://doi.org/10.1109/JRPROC.1961.287814

Mattamorphic. (2023). Cyclic Redundancy Check. Mattamorphic GitHub