UNIVERSIDAD DE CONCEPCIÓN

FACULTAD DE INGENIERÍA

DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

Profesores Patrocinantes:
Guía: Dr. Miguel Figueroa
Co-Guía: Dra. Cecilia Hernández

Informe de Tesis para optar al grado de:
Magíster en Ciencias de la Ingeniería con mención en Ingeniería Eléctrica

Acelerador Hardware para la Estimación de Entropía Empírica Mediante Sketches

Concepción, Agosto de 2021

Paulo Pedro Ubisse

Universidad de Concepción
Facultad de Ingeniería
Departamento de Ingeniería Eléctrica

Profesores Patrocinantes:
Guía: Dr. Miguel Figueroa
Co-Guía: Dra. Cecilia Hernández

Acelerador Hardware Para La Estimación De Entropía Empírica Mediante Sketches

Paulo Pedro Ubisse

Informe de Tesis
para optar al grado de

"Magíster en Ciencias de la Ingeniería con mención en Ingeniería Eléctrica"

Agosto 2021

Resumen

Los enfoques basados en entropía para la detección de anomalías son atractivos, ya que proporcionan información más detallada que el análisis de volumen de tráfico tradicional. Sin embargo, el cálculo de la entropía empírica exacta en un gran conjunto de datos puede ser costoso en utilización de memoria, debido a que su cómputo requiere almacenar el número de ocurrencias de todos los elementos distintos observados en el flujo. Un alto uso de memoria reduce el desempeño de los aceleradores hardware, que son necesarios para estimar la entropía en redes de alta velocidad. Una solución práctica es, entonces, relajar la restricción de un cálculo exacto.

En este trabajo, presentamos dos métodos probabilísticos basados en sketches para aproximar la entropía empírica de un gran conjunto de datos en procesamiento en tiempo real, con uso de espacio de memoria sublineal. Los sketches son estructuras de datos que utilizan espacio sublineal donde el uso de memoria crece de forma sublineal con los datos de entrada. Cuando el tamaño de la memoria utilizada es menor que la entrada, la pérdida de precisión es inevitable y conduce a resultados probabilísticos. Sin embargo, los algoritmos basados en sketches proporcionan aproximaciones con resultados de alta calidad. El primer enfoque consiste en estimar la entropía empírica de un flujo de datos, considerando los elementos top-K, o sea los K elementos más frecuentes. El segundo y principal enfoque de nuestro trabajo, consiste en aproximar la estimación de la entropía empírica no solo tomando la parte del flujo de datos que corresponde a los K elementos más frecuentes, sino también tomando la parte del flujo de datos que corresponde a los elementos menos frecuentes.

Los dos enfoques han sido implementados en un sistema en chip (System on chip, SoC) Zynq-UltraScale de Xilinx. Para el primer enfoque, los resultados experimentales del diseño de hardware en una FPGA Xilinx Zynq UltraScale+MPSoC ZCU102 (*Field Programmable Gate Array*), muestran que el sistema funciona a una frecuencia de reloj de 354 MHz y puede funcionar con una velocidad de red de 181 Gigabits por segundo (Gbps). Para el segundo y principal enfoque, los resultados experimentales en una arquitectura de propósito especial implementada en un FPGA ZCU104 de ultraescala Xilinx Zynq, muestran que el sistema alcanza un rendimiento de un paquete por ciclo a 400 MHz, lo que le permite operar a velocidades de red de hasta 204 Gbps.

"Los mejores libros son los que nos dicen lo que ya sabemos."

- Winston.

Agradecimientos

En primer lugar, agradezco al Señor mi Dios todo poderoso por guiarme en este largo viaje. Gracias por el don de la vida Señor.

Mi especial agradecimiento a mis profesores guías, que me han ayudado en esta larga caminada, en especial al profesor Miguel Figueroa y la profesora Cecilia Hernández, por su disponibilidad, ayuda, dedicación y orientación durante la elaboración de este trabajo, así como por su confianza en mí. Muchas gracias!

Al Javier Soto y Yaime por toda la ayuda y aportes para el desarrollo de este trabajo. Gracias también para mis amigos Felix Carmona, Gizela Muege, Gonçalves Massingue, Afonso Zunguza, Alfaid Jone quienes me animaron durante la elaboración de este proyecto. Mi Fuerte Abrazo. Agradezco igualmente la Maria Ignacia por toda la ayuda y disposición durante la elaboración de este trabajo.

También agradezco a mis padres por quienes tengo un cariño inconmensurable y que fueron un gran suporte durante mi recorrido académico, al mi eterno padre Pedro Ubisse, y a las dueñas Albertina Matavele, y Júlia Cambaco. También me gustaría agradecer a todos mis compañeros del laboratório de VLSI que fueron una gran familia, especialmente a los compañeros Bárbaro, Wladimir, Ignacio Perez y Pablo Verdugo. Mil gracias (Khanimambo)!

Durante las diferentes etapas de este trabajo he recibido el apoyo financiero del Programa de Becas para Estudios de Magíster en Chile de la AGCID y FONDECYT del Gobierno de Chile.

Tabla de Contenidos

Agradecimientos iii

Lista de Figuras vii

Lista de Tablas viii

Capítulo 1 Introducción 1
1.1 Introducción general . . . 1
1.2 Estado del arte . . . 2
1.2.1 Implementaciones en hardware de algoritmos de sketch . . . 3
1.2.2 Algoritmos de detección de anomalías de tráfico basados en entropía empírica . . . 5
1.3 Motivación . . . 6
1.4 Hipótesis . . . 6
1.5 Objetivos . . . 7
1.5.1 Objetivo general . . . 7
1.5.2 Objetivos específicos . . . 7
1.6 Temario . . . 7

Capítulo 2 Fundamentos teóricos 8
2.1 Algoritmos de streaming de datos . . . 8
2.2 Algoritmos de sketches . . . 8
2.3 Problemas de elementos frecuentes . . . 9
2.3.1 Estructura de sketches para el conteo de elementos . . . 10
2.4 Problemas de elementos distintos . . . 11
2.4.1 Estructura de sketches para el conteo de elementos distintos . . . 12
2.4.2 *Probabilistic Counting with Stochastic Averaging* . . . 13
2.4.3 *LogLog* y *Super-LogLog* . . . 13
2.4.4 *HyperLogLog* . . . 15
2.4.5 Discusión . . . 16
2.5 Entropía empírica . . . 18
2.5.1 Definición matemática de la entropía . . . 18

Capítulo 3 Metodología 19
3.1 Restricciones en el cálculo real de la entropía . . . 19
3.2 Enfoques propuestos para estimar la entropía . . . 19
3.2.1 Enfoque 1 . . . 20
3.2.2 Enfoque 2 . . . 20
3.3 Algoritmos . . . 22
3.3.1 Enfoque 1 . . . 22
3.3.2 Enfoque 2 . . . 24
3.3.3 Algoritmos para estimación de ocurrencias . . . 25
3.3.4 Algoritmos para estimación de cardinalidad . . . 26
3.3.5 Función Hash . . . 27

Capítulo 4 Arquitectura del sistema 28
4.1 Arquitecturas de hardware . . . 28
4.1.1 Arquitectura aceleradora de hardware del enfoque 1 . . . 28
4.1.2 Arquitectura aceleradora de hardware del enfoque 2 . . . 28
4.1.3 Módulo Countmin-CU sketch . . . 29
4.1.4 Módulo MurmurHash3 . . . 31
4.1.5 Módulo Cola de prioridad . . . 31
4.1.6 Módulo Hyperloglog . . . 34

4.1.7 Módulo de estimación de entropía 35

Capítulo 5 Resultados **36**
5.1 Enfoque 1 36
5.1.1 Base de datos usadas 36
5.1.2 Resultados del enfoque 1 37
5.2 Enfoque 2 43
5.2.1 Base de datos usadas 43
5.2.2 Resultados del enfoque 2 44

Capítulo 6 Conclusiones y trabajos futuros **54**
6.1 Conclusiones 54

Referencias **61**

Lista de Figuras

2.1 Sketch genérico para el conteo de elementos. 11
2.2 Sketch genérico para estimación de elementos distintos. 15

4.1 Arquitectura general del sistema según el enfoque de la sección 3.2.1. 28
4.2 Arquitectura general del sistema según el enfoque de la sección 3.2.2. 29
4.3 Arquitectura CM-CU sketch según el enfoque de la sección 3.2.1.[1] 29
4.4 Arquitectura CM-CU sketch según el enfoque de la sección 3.2.2.[2] 30
4.5 Arquitectura del módulo de cola de prioridad.[1] 32
4.6 Circuito de actualización para la cola de prioridad.[2] 33
4.7 Arquitectura del módulo Hyperloglog.[2] 34
4.8 Bloque básico del "1" más significativo.[1] 35

5.1 Error relativo de estimación de entropía según la ecuación (3.6) con respecto a la ecuación (2.9) en función de K, considerando los valores exactos de N y m_i. . 45
5.2 Error relativo con respecto a la estimación de entropía según la ecuación (3.6) cuando se usa un HLL sketch para estimar N, en función del parámetro de precisión p. 46

Lista de Tablas

5.1 Información detallada de las trazas de tráfico de red utilizados en los experimentos. 37
5.2 Estimación de la entropía normalizada y su error absoluto E_{est}, en función de K usando el enfoque dado por la ecuación (3.2). 38
5.3 Estimación de la entropía normalizada y su error relativo E_{r_est}, en función de K usando el enfoque dado por la ecuación (3.2). 38
5.4 Reporte de la estimación de la entropía, error absoluto y error relativo de los top-10240, según el primer enfoque 39
5.5 Error absoluto de estimación de entropía utilizando la ecuación (3.2) como referencia, para tres tipos de sketches y seis tamaños. 40
5.6 Reporte del error absoluto de estimación de la entropía considerando el sketch y arreglo de colas. 41
5.7 Reporte de la estimación de la entropía usando el acelerador. 42
5.8 Utilización de recursos FPGA por módulo 42
5.9 Información detallada de las trazas de tráfico de red utilizados en los experimentos. 44
5.10 Error relativo de estimación de entropía utilizando la ecuación (3.6) como referencia, para tres tipos de sketches y cuatro tamaños. 48
5.11 Resumen de la estimación de entropía obtenida por la ecuación (3.6) y el acelerador. 50
5.12 Resumen de los errores de estimación absolutos y relativos obtenidos por la ecuación (3.6) y el acelerador. 51
5.13 Utilización de recursos FPGA por módulo 52
5.14 Consumo de energía FPGA por módulo 53

Capítulo 1. Introducción

1.1. Introducción general

En los últimos años se ha verificado un cambio en el enfoque del análisis de tráfico de red basado en el volumen para el análisis basado en la distribución del flujo de red, debido a que, este puede capturar el estado de la red de manera más sucinta [3]. Uno de los principales desafíos que se enfrentan en este contexto es el alto costo computacional asociado con la detección de eventos en tiempo real. De acuerdo con [4], detectar eventos anómalos en redes de IP rápidas, como redes troncales de Internet, es difícil. Un problema es que la cantidad de datos de tráfico no permite el análisis de detalles en tiempo real. Otro es que las características específicas de estos eventos no se conocen con anterioridad. Esto genera una necesidad de métodos de análisis que sean capaces de ejecutar procesamiento en línea de grandes cantidades de datos.

Muchas veces, el procesamiento de grandes volúmenes de datos no requiere soluciones exactas, siendo suficiente encontrar aproximaciones que se puedan calcular de manera mucho más eficiente. Esta estrategia conocida como computación aproximada, se ha utilizado en ciencias computacionales durante años, aplicándose en aquellos contextos donde las respuestas que están lo suficientemente cerca del valor actual son aceptables. Esto da lugar a una compensación de precisión por otros recursos, típicamente espacio de memoria y tiempo.

Una de las técnicas más llamativas para acelerar el procesamiento de grandes cantidades de datos, es el uso de algoritmos de streaming de datos basados en sketches. Estos pueden administrar la memoria y el tiempo de manera efectiva y eficiente para manejar grandes volúmenes de datos en muchos campos de aplicaciones [5]. Estos han sido utilizados donde muchas veces no es posible su almacenamiento, pero se requiere tomar decisiones en línea conforme se leen los datos (haciendo uno o número reducido de pasos).

Los sketches son estructuras de datos que utilizan un espacio sublineal, lo que significa que el tamaño de la memoria utilizada crece de forma sublineal con los datos de entrada [6, 7]. Cuando el tamaño de la memoria utilizada es más pequeño que la entrada, puede ocasionar una pérdida de precisión, lo que lleva a resultados probabilísticos [8]. Sin embargo, los algoritmos basados en sketches proporcionan aproximaciones con resultados de alta calidad, lo que comúnmente es tan útil como los resultados exactos. Además, permite la implementación en arquitecturas de aceleración de hardware basadas en FPGA. Las arquitecturas de hardware dedicadas son

importantes en aplicaciones que requieren un alto rendimiento de datos porque pueden procesar los datos en tiempo real con un rendimiento predecible a un costo y consumo de energía más bajos que las arquitecturas programables tradicionales [1].

En este trabajo, presentamos dos enfoques para estimar la entropía de las direcciones IP utilizando trazas de red estándar. El primer enfoque utiliza un algoritmo de streaming de datos basado en sketches para contar elementos frecuentes. La idea es de estimar la entropía considerando los K elementos más frecuentes. El segundo y principal enfoque utiliza dos algoritmos de streaming de datos basados en sketches. Uno para contar elementos frecuentes y otro para contar la cantidad de elementos distintos. Este último enfoque usado para estimar la entropía, considera tanto los K elementos más frecuentes, como los elementos menos frecuentes, asumiendo que estos siguen una distribución uniforme. Los resultados muestran que el principal enfoque de este trabajo proporciona una buena estimación de la entropía de toda la secuencia, con un error relativo máximo menor a 3 % sobre trazas de redes de tráfico de datos.

1.2. Estado del arte

El concepto de entropía se ha utilizado en aplicaciones de monitoreo de redes como un enfoque para detectar cambios repentinos en el comportamiento de la red y como indicador de eventos anómalos [9, 10, 11, 12]. Además, es importante en una amplia gama de áreas de aplicación, como aprendizaje automático [13, 14, 15], minería de datos [16, 17, 18, 19], análisis de registros de búsqueda comerciales [20] y procesamiento de señales [21, 22]. En el contexto del monitoreo del tráfico de red con el propósito de detección de anomalías o agrupamiento de tráfico, el concepto de entropía es una métrica esencial debido a que brindan información más detallada que las métricas tradicionales basadas en volumen no pueden identificar [23].

La complejidad radica en el hecho de que calcular la entropía para todos los elementos en un gran flujo de datos, desde un punto de vista computacional, es costoso. Primero, debido a que no es factible mantener un contador para cada elemento distinto, no hay suficientes recursos de memoria para almacenar todos los elementos del flujo de datos. En segundo lugar, porque las anomalías suelen detectarse en tiempo real. Esto es difícil porque los enlaces actualmente operan a altas velocidades, más de 100 Gigabits por segundo (Gbps) [24, 25]. Sin embargo, muchos estudios muestran que procesar grandes cantidades de datos no siempre requiere respuestas exactas, sino que es suficiente encontrar una buena aproximación [26, 27, 28]. Esto se puede lograr mediante el uso de algoritmos aproximados. Estos permiten la gestión de la memo-

ria y el tiempo de manera efectiva y eficiente y proporcionan resultados aproximados con bajo error [26, 8, 5], resolviendo así el primer punto de limitación. El uso de algoritmos de streaming con memoria sublineal permite el uso de aceleradores de hardware para poder lograr un rendimiento en tiempo real, resolviendo así la segunda limitación del procesamiento de datos en tiempo real. Las arquitecturas de hardware dedicadas tienen más ventajas que la CPU (*Central Processor Unit*), porque son capaces de realizar procesamiento paralelo mientras mantienen un bajo consumo de energía en sistemas embebidos [29, 30].

1.2.1. Implementaciones en hardware de algoritmos de sketch

En [31], los autores proponen una arquitectura genérica en FPGA para acelerar con hardware algoritmos en línea basados en sketch para dos tareas clave de detección de anomalías de red: detección de heavy hitters y detección de cambios intensos. La arquitectura propuesta para acelerar los sketches usa CountMin sketch para detección de heavy hitters, está implementado en un FPGA Xilinx Virtex Ultrascale XCVU440. La velocidad de reloj máxima alcanzada es de 477 MHz. Los mejores rendimientos fueron alcanzados con los sketches implementados con dimensiones de 32,768 x 5 ($w = 2^{15}$, $d = 5$). Con esta arquitectura del hardware se puede operar en enlaces de 150 Gbps para detección de heavy hitters, 100 Gbps para detección de heavy change y se logra una mejora del rendimiento de 2-400x en comparación con otras técnicas de acceleración descritas en el paper.

En [32], los autores proponen una arquitectura basada en FPGA para clasificar los elementos en un flujo de datos (clasificación de las direcciones IP de origen para encontrar heavy hitters en un enlace de red, es decir, encontrar aquellas direcciones IP que generan más paquetes). En este se presenta la integración del algoritmo Count Sketch con una lista de prioridades y un analizador de paquetes de red, que está implementado con éxito en un FPGA Xilinx Virtex Ultrascale VCU108. La velocidad de reloj máxima alcanzada es de 322 MHz. Las dimensiones del sketch implementado son de 1,048,576 x 8 ($w = 2^{20}$, $d = 8$), con un error inferior al 0,1 %. Con esta arquitectura, es capaz de procesar más de 26,855,731 paquetes por segundo en enlaces de 100 Gbps para la detección de heavy hitters. La implementación ha sido validada utilizando trazas del mundo real, obteniendo un error promedio de 1.29 %. A diferencia de otras soluciones, el estimador utilizado en este trabajo no está sesgado, no requiere múltiples inspecciones de los datos entrantes y no utiliza el muestreo.

En [33], los autores proponen un algoritmo de sketch híbrido que descompone la detección jerárquica de heavy hitter en línea en una detección de heavy change en línea independiente y paralela

en cada nivel de jerarquía. Además, se propone una arquitectura segmentada adecuada para la implementación en FPGA para acelerar el algoritmo. La arquitectura está implementada en un FPGA Xilinx Virtex-7 XC7VX690. La velocidad máxima de reloj alcanzada es de 384 MHz. Los mejores resultados son obtenidos con dimensiones del sketch de 65,536 x 5 ($w = 2^{16}$, $d = 5$). Con esta arquitectura, se logra alcanzar un rendimiento de 123 Gbps. El consumo de recursos es de: 20.51 % de bloques BRAM, 20.91 % de LUTs y 16.05 % de registros.

En [34], los autores proponen un Módulo de Extracción de Características (*FEM-Feature Extraction Module en inglés*), exacto y escalable basado en sketches. Además, presentan los detalles del diseño de una FEM en un FPGA y demuestran que usando FPGA se puede lograr un rendimiento significativamente mejor en comparación con las implementaciones existentes en software y las implementaciones ASIC. La arquitectura está implementada en un FPGA Xilinx Virtex-II XC2V1000. Este opera a una frecuencia de reloj de 270 MHz. Con está arquitectura se logra aumentar la precisión hasta 97.61 %, reducir el error de estimación a un promedio de 0.0365 paquetes y rendimientos de hasta 21.25 Gbps para un FEM de entrada de 16K. En el trabajo citado en [33], los autores proponen una arquitectura de cómputo de hash de alto rendimiento, optimizan el algoritmo de CountMin sketch para la detección de los heavy hitters en línea por hardware. La arquitectura utiliza el countmin sketch que fue implementado en un FPGA Xilinx Virtex-7 XC7VX1140. La velocidad máxima de reloj alcanzada es de 384 MHz. Las dimensiones del sketch implementado son de 65,536 x 5 ($w = 2^{16}$, $d = 5$). Con esta arquitectura se logra un rendimiento de 114 Gbps.

Los autores en [35], proponen una arquitectura de propósito especial para la detección de heavy hitter. Se ha probado su rendimiento en el tráfico de red y los conjuntos de datos Chip-seq. La arquitectura utiliza el algoritmo de Countmin-CU sketch que fue implementado en un FPGA Xilinx Kintex-7 XC7K325T. Las dimensiones del sketch son de 16,384 x 4 ($w = 2^{14}$, $d = 4$). El diseño implementado logra un rendimiento de hasta un elemento de datos por ciclo de reloj, con una latencia inicial de 25 ciclos y una frecuencia de reloj de 300 MHz. Este explota el paralelismo de datos en el algoritmo en dos niveles: en primer lugar, cada fila del sketch se administra de forma independiente y en paralelo mediante funciones de hash independientes y circuitos de acceso a la memoria. La matriz se implementó utilizando bloques de memoria separados para cada fila, lo que permite lecturas y actualizaciones de contadores paralelos. En segundo lugar, los circuitos que computan las funciones hash, los incrementos del contador, la selección del valor mínimo y las actualizaciones de la memoria fueron totalmente canalizados para maximizar la frecuencia de reloj del sistema.

1.2.2. Algoritmos de detección de anomalías de tráfico basados en entropía empírica

En [3], los autores investigan el problema de estimar la entropía en un modelo de cálculo de streaming, dando límites más bajos para este problema, mostrando que ni la aproximación ni la aleatorización por sí solas nos permitirán calcular la entropía de manera eficiente. De esta forma, como solución, presentan dos algoritmos para aproximar aleatoriamente la entropía de una manera eficiente en el tiempo y el espacio, aplicable para su uso en enlaces de muy alta velocidad (mayor que OC-48 *"Optical Carrier 48"*). En el primer algoritmo, se proporcionan fuertes garantías teóricas sobre el error y el uso de recursos, está inspirado en la similitud estructural con el trabajo seminal de Alon et al. para estimar momentos de frecuencia. El segundo algoritmo utiliza la observación de que el rendimiento del algoritmo de streaming se puede mejorar separando los elementos de alta frecuencia de los elementos de baja frecuencia.

En [36], los autores basados en la teoría de que utilizando una submuestra aleatoria, pueden calcular una aproximación de la entropía empírica de manera eficiente y derivan los límites de error probabilísticos para la aproximación, donde los límites de error se reducen en una tasa de raíz cuadrada cercana con respecto al tamaño de la submuestra. Además, presentaron dos aplicaciones que pueden beneficiarse de la aproximación limitada por errores: clasificación de características y filtrado basado en información mutua. De esta forma, como solución, han desarrollado algoritmos para submuestrear progresivamente el conjunto de datos y devolver respuestas correctas con alta probabilidad.

Los autores en [37], presentan una metodología general para crear perfiles de comportamiento integrales del tráfico troncal de Internet en términos de patrones de comunicación de los *hosts* finales y los servicios. Basándose en minería de datos y técnicas basadas en entropía, la metodología consiste en la extracción significativa de clústeres, la clasificación automática del comportamiento y el modelado estructural para análisis interpretativos en profundidad. Utilizan minería de datos y técnicas basadas en entropía para descubrir automáticamente patrones de comportamiento significativos a partir de datos de tráfico a nivel de enlace, y para proporcionar interpretaciones plausibles de los comportamientos observados. También, demuestran la aplicabilidad del enfoque de perfil para el problema de detección de tráfico no deseado y anomalías. Además investigan posibles estrategias de contramedidas que un provedor de servicios de internet (ISP), de red troncal puede seguir para reducir el tráfico de fuentes no deseadas en función de sus características. Los resultados demostraron que el bloqueo de las fuentes más infractoras es razonablemente rentable.

1.3. Motivación

Lo que impulsa este trabajo es que muchos estudios han sugerido el uso de la entropía como un medio sucinto de resumir las distribuciones de tráfico para diferentes aplicaciones, en particular, en la detección de anomalías y en el análisis y clasificación de tráfico de grano fino. Con respecto a la detección de anomalías, el uso de la entropía para rastrear los cambios en las distribuciones de tráfico proporciona dos beneficios significativos. Primero, el uso de entropía puede aumentar la sensibilidad de detección para descubrir incidentes anómalos que pueden no manifestarse como anomalías basadas en el volumen. En segundo lugar, el uso de tales funciones de tráfico proporciona información de diagnóstico sobre la naturaleza de los incidentes anómalos (por ejemplo, haciendo distinción entre gusanos, ataques DDoS y escaneos) que no está disponible solo mediante la detección de anomalías basada en el volumen. Con respecto al análisis de tráfico detallado y la clasificación del tráfico, la entropía de las distribuciones de características de tráfico ofrece información útil para medir la distancia entre los grupos (de tráfico).

Por otro lado, el cálculo de la entropía empírica en un conjunto de datos a gran escala puede ser costoso, debido a que, en la mayoría de las aplicaciones, el conjunto de datos es demasiado grande para mantenerse en la memoria central. Sin embargo, podemos calcular una aproximación de la entropía empírica de manera eficiente, dado que, en muchas aplicaciones prácticas, es aceptable un error relativo menor al 5 %. De esta forma, pretendemos estimar la entropía empírica utilizando algoritmos probabilísticos basados en sketches. Los sketches son estructuras de datos que utilizan un espacio sublineal, lo que significa que el tamaño de la memoria utilizada crece de forma sub-lineal con los datos de entrada. Estos proporcionan aproximaciones de alta calidad, lo que comúnmente es tan útil como los resultados exactos. Además, son fáciles de implementar en arquitecturas de aceleración de hardware basadas en FPGA.

1.4. Hipótesis

En este trabajo se postula la posibilidad de realizar una implementación eficiente, en hardware de lógica programable y de alto desempeño, de algoritmos basados en sketches, para la estimación de entropía empírica en un gran flujo de datos. Esta implementación, además de eficiente, garantiza una alta tasa de procesamiento y bajo consumo de potencia, para su utilización en procesos de alta exigencia computacional y en tiempo real.

1.5. Objetivos

1.5.1. Objetivo general

El objetivo de este trabajo es realizar el diseño de una arquitectura aceleradora de hardware para la estimación de entropía empírica en un grande flujo de datos, utilizando algoritmos basados en sketches. Éste será implementado en una tarjeta de desarrollo con lógica programable.

1.5.2. Objetivos específicos

1. Plantear y analizar una solución de aproximación del cálculo de entropía empírica.
2. Identificar y analizar los requerimientos de implementación de sketches.
3. Formular estrategias de diseño de aceleradores hardware de algoritmos de estimación de entropía empírica para alcanzar alto desempeño con un uso eficiente de recursos.
4. Diseñar, implementar y validar aceleradores hardware para la estimación de la entropía empírica, utilizando sketches.

1.6. Temario

El temario de este trabajo se describe a continuación:

- El Capítulo 2 presenta una revisión teórica de los algoritmos utilizados en este trabajo.
- El Capítulo 3 describe la metodología utilizada para evaluar el desempeño de los algoritmos y los resultados obtenidos en la pruebas en software realizadas.
- El Capítulo 4 presenta el desarrollo de la implementación del sistema en un sistema embebido, incluyendo la arquitectura y las consideraciones de diseño.
- El Capítulo 5 muestra los resultados obtenidos en cada parte del sistema.
- El Capítulo 6 recopila las conclusiones alcanzadas y propone el trabajo a futuro.

Capítulo 2. Fundamentos teóricos

2.1. Algoritmos de streaming de datos

Los algoritmos de streaming de datos procesan y realizan cálculos en flujos de datos. En el modelo de cálculo de flujo de datos, la entrada se presenta como elementos que llegan secuencialmente con repeticiones [38]. Normalmente, los algoritmos de transmisión tienen un espacio de memoria limitado en relación con la longitud del flujo de datos y el tiempo de procesamiento por elemento. Otra restricción en las aplicaciones de monitoreo de tráfico de red de alta velocidad es que los elementos (paquetes) en el flujo de datos (tráfico de red) deben examinarse en una pasada porque es imposible almacenar paquetes debido al espacio de memoria limitado en los dispositivos de red. Por lo tanto, dichos algoritmos emplean estructuras de datos de sinopsis como la estructura de datos de sketches [39].

2.2. Algoritmos de sketches

Dada una entrada de n ítems $(x_1, x_2, \ldots, x_n)$, cada ítem x, se mapea a través de funciones *hash* en un pequeño vector de sketch que registra información de frecuencia. Por lo tanto, el sketch no almacena los elementos explícitamente, sino solo información sobre la distribución de frecuencias. Los sketches admiten consultas fundamentales en su entrada, como consultas de punto, rango y producto interno para que se responda rápidamente.

Los algoritmos basados en sketches realizan un resumen, principalmente mediante hashing y conteo, para una estadística de interés específica. Han ganado mucho interés en la comunidad de investigación en los últimos años. Debido a su complejidad de memoria sublineal, las técnicas basadas en sketches consumen significativamente menos memoria que las técnicas tradicionales por elemento de estado para procesar flujos de datos de alto rendimiento. Por lo tanto, esta baja complejidad de la memoria, permite con que las técnicas basadas en sketches pueden ser compatibles con el rápido almacenamiento en el chip de las plataformas de computación de vanguardia para lograr un alto rendimiento.

Los sketches están siendo utilizados en un amplio espectro de aplicaciones en escenarios de *Big Data*. Estas aplicaciones incluyen compartir cachés web [40], rastrear flujos en el tráfico de la red

[41, 42, 43], detectar ataques en los enrutadores [44], detectar anomalías en la red[45], encontrar elementos frecuentes (tales como heavy hitters) [46, 47, 48], agregar estadísticas en redes de sensores [49] y procesar conjuntos de datos distribuidos [50, 51, 52]. Las grandes corporaciones de redes como AT&T [53] y Google [54] también usan sketches para el monitoreo de redes y la administración de datos de redes. Son cada vez más dispositivos que se conectan a Internet y causan un aumento en la cantidad de datos de Internet a una tasa sin precedentes (el tráfico total de Internet superará los 1 ZB en 2016 y aumentará a 2.3 ZB para 2020 [55]).

Como los sketches se utilizan a menudo en configuraciones donde se consultan a una alta velocidad, deben implementarse en la memoria rápida integrada al chip. Esta es una memoria rápida que puede leerse y escribirse muy rápido, como las BRAM en chips FPGA. Desafortunadamente, la memoria rápida es costosa y de tamaño limitado, por lo tanto, a menudo se usa para almacenar estructuras de datos compactas con requisitos de alta velocidad. Por el contrario, la memoria externa al chip es una memoria lenta cuya velocidad de lectura y escritura es aproximadamente diez veces más lenta que la de la integrada al chip (como la RAM Dinámica, DRAM*"Dynamic Random Access Memory"*) [56].

2.3. Problemas de elementos frecuentes

Muchas veces la acción de encontrar los elementos top-K se puede resolver mediante técnicas basadas en el contador. Sin embargo, se ha notado que el uso de un contador particular para cada candidato posible en el flujo de datos, y la clasificación de los diferentes contadores por valor, es una solución eficiente cuando los requisitos del problema están bien definidos.

Uno de los principales inconvenientes de esta aproximación es que el número total de contadores aumenta linealmente con el número de elementos monitoreados. Otro inconveniente es que si el número de contadores se define estáticamente en el momento del diseño y se observa un elemento inesperado, el diseño descartará su información asociada.

La principal crítica asociada a los enfoques basados en el contador es el uso intensivo de la memoria. Este inconveniente hace que sea muy difícil aplicar estas técnicas en un entorno del mundo real donde los elementos del flujo de datos de entrada pueden pertenecer a un alfabeto muy diverso. Alternativamente, el muestreo de los elementos entrantes es una solución explotada por dispositivos de red comerciales.

A medida en que se detectaba el aumento en el número de elementos y sus frecuencias en un

conjunto múltiple, se introdujo técnicas basadas en sketches. Las técnicas basadas en sketches consumen significativamente menos memoria que las técnicas tradicionales por elemento de estado para procesar flujos de datos de alto rendimiento.

De acuerdo con [57], un sketch es una estructura de datos probabilística que almacena las frecuencias de los elementos en un conjunto múltiple y devuelve una estimación de la frecuencia de cualquier elemento dado cuando se solicita. Un conjunto múltiple es un conjunto en el que un elemento puede aparecer más de una vez.

Los sketches se están utilizando ampliamente en un gran número de aplicaciones del mundo real para estimar las frecuencias de los elementos de datos. Debido al aumento sin precedentes en la cantidad de datos de Internet y un aumento relativamente más lento en el tamaño de las memorias integradas al chip, los sketches existentes son cada vez más incapaces de mantener la precisión de las estimaciones de frecuencia a un nivel aceptable.

La razón es que con el aumento en el número de elementos y sus frecuencias en un conjunto múltiple, los sketches necesitan más memoria para almacenar y estimar con precisión las frecuencias. Como los sketches se utilizan a menudo en configuraciones donde se consultan a una alta velocidad, deben implementarse en la memoria rápida integrada al chip. Desafortunadamente,la memoria integrada al chip es muy limitada. Por lo tanto, con la creciente cantidad de datos, los sketches existentes no están obteniendo suficiente memoria integrada al chip para almacenar las frecuencias con precisión.

2.3.1. Estructura de sketches para el conteo de elementos

Para estimar la ocurrencias de elementos e del flujo de red, se presenta los algoritmos de flujos de datos existentes basadas en sketches más usados para resolver el problema de cuenta de elementos: CountMin (CM) [58], CountMin con actualizaciones conservadoras (CM-CU) [59] y CountSketch (CS) [46]. Los tres algoritmos utilizan la estructura de datos que se muestra en la Fig. 2.1.

La estructura presentada en la Figura 2.1, se caracteriza como una matriz C de contadores con d filas y w columnas, donde cada celda almacena un contador de frecuencia. Los tres algoritmos usan d funciones hash para mapear un elemento e de entrada en una posición de columna en C para cada una de las d filas. La operación *update(e)* difiere ligeramente en los tres algoritmos de sketch: CM incrementa en uno de los d contadores seleccionados en C, CM-CU solo incrementa

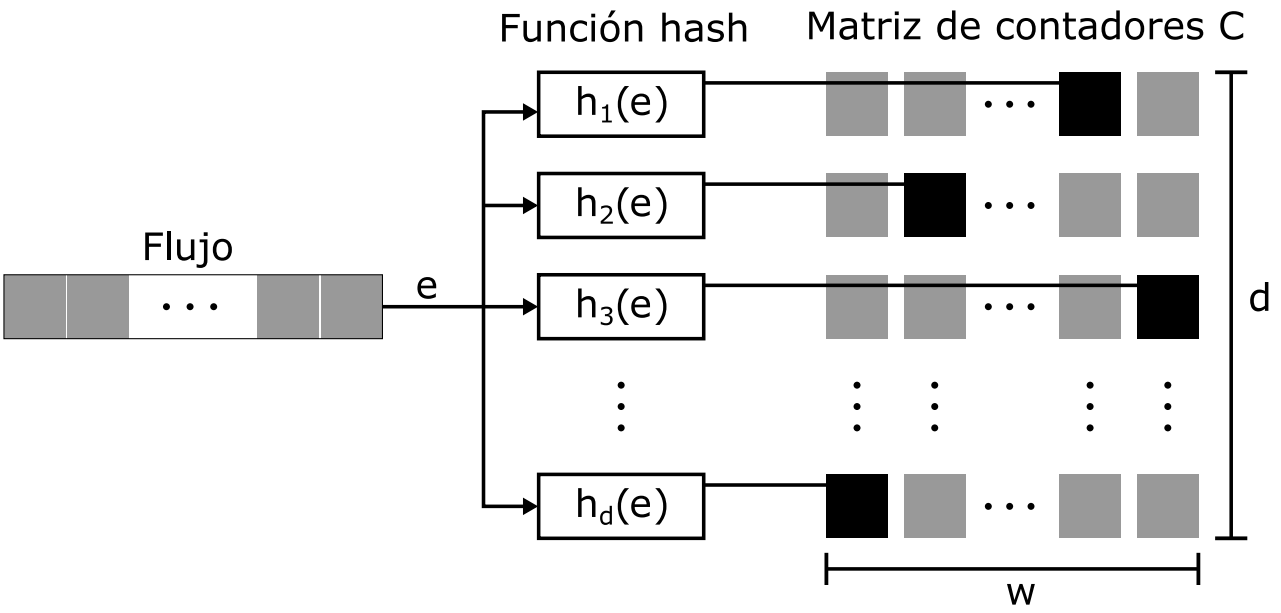

Figura 2.1: Sketch genérico para el conteo de elementos.

los contadores que contienen el valor mínimo y CS usa d funciones hash adicionales para decidir si se incrementa o decrementa cada contador seleccionado. La operación *estimación(e)* para CM y CM-CU devuelve el contador con el valor mínimo, mientras que el CS devuelve el valor mediano de los contadores seleccionados d. El CS puede subestimar o sobreestimar las frecuencias de los elementos, mientras que CM y CM-CU solo pueden sobreestimar [58].

Goyal et al. [60] y Basat et al. [61] han comparado CM con CM-CU y han demostrado que CM-CU puede, en la práctica, reducir la sobreestimación en comparación con CM. Xiao et al. [62] evaluaron CM-CU y encontraron que produce mejores estimaciones de frecuencia que CM y CS. En [35, 63], se ha mostrado que, al identificar a los heavy hitters en un conjunto de datos, CM-CU produce resultados más precisos con menos uso de memoria en comparación con CM y CS.

2.4. Problemas de elementos distintos

La determinación del número de elementos distintos es un tema importante en muchas aplicaciones de red. Estas aplicaciones incluyen compartir cachés web [40], rastrear flujos en el tráfico de la red [41, 42, 43], detectar ataques en los enrutadores [44], detectar anomalías en la red [45], encontrar elementos frecuentes (tales como *heavy hitters*) [46, 47, 48], agregar estadísticas

en redes de sensores [49] y procesar conjuntos de datos distribuidos [50, 51, 52], etc. Muchas aplicaciones de base de datos, como la optimización de consultas de base de datos, requieren una estimación rápida y precisa.

Dado un *multiset* $S = (x_1, x_2, ...)$ cuyo tamaño denominaremos $M = |S|$ (número total de elementos en S) se está interesado en obtener el número de elementos distintos que contiene S, denominado n. Este problema también es conocido como *cardinality* y su complejidad radica en que, para *multisets* donde N y M son significativamente grandes, el espacio necesario para conseguir el valor de m puede ser demasiado elevado. En el tipo de problemas para el cual se utilizan este tipo de algoritmos habitualmente no es necesario saber el valor N exacto, sino una aproximación de la cual podamos obtener una idea de su magnitud. Para ello, debemos utilizar un tiempo lineal a la entrada y una memoria sublineal, típicamente *O(log N)* o *O(log log N)*.

La estimación de cardinalidad es útil en temas relacionados con la *World Wide Web* (WWW). La WWW es un sistema de distribución de documentos en el que interactúan millones de ordenadores a diario. Para que todo funcione se usan una serie de protocolos, como por ejemplo HTTP (*Hypertext Transfer Protocol*), que envía una petición al servidor solicitando un recurso. La primera parte para desarrollar con éxito esa petición es la traducción de la URL (*Uniform Resource Locator*), en una dirección IP. Además, los paquetes de datos que enviamos a través de la red también estarán identificados en sus respectivas cabeceras con una dirección IP. Para controlar y medir todo ese tráfico es habitual utilizar este tipo de algoritmos [64]. La complejidad espacial de contar el número exacto de elementos distintos en una secuencia es lineal en la cardinalidad del conjunto o el tamaño de la secuencia. Por lo tanto, para estimar la cardinalidad con memoria limitada, se han propuesto varios algoritmos de transmisión basados en sketches [65, 66, 67]. Entre estos, el HyperLogLog (HLL) tiene una complejidad de espacio sublineal y proporciona un error de estimación bajo en teoría [67] y en la práctica [68]. Para el presente trabajo, usamos el HLL sketch para estimar el número de elementos distintos N en un flujo de red.

2.4.1. Estructura de sketches para el conteo de elementos distintos

Para que se pueda llevar a cabo la estimación de cardinalidad hay que hacer el análisis de algunos tipos de algoritmos (Flajolet-Martin PCSA-*"Probabilistic Counting with Stochastic Averaging"*, *LogLog*, *Super-LogLog* y *HyperLogLog*).

2.4.2. *Probabilistic Counting with Stochastic Averaging*

Nigel Martin y Philippe Flajolet son dos de los pioneros en los algoritmos probabilísticos de flujo, siendo Flajolet el que ha tenido una mayor contribución en el tema. En el año 1978, Martin trabajaba para IBM y desarrolló lo que podrían considerarse el primer conjunto de ideas que darían paso a la primera versión de *Probabilistic Counting* [65].

En el conteo probabilístico, se busca obtener una estimación razonable de varios elementos únicos. Si por ejemplo, al asumir que se tiene una cadena de longitud m que consiste en $\{0, 1\}$ con igual probabilidad. La probabilidad de que esta comience en 1, con 2 unos, con k unos, es $1/2$, $1/4$ y $1/2^k$ respectivamente. Esto significa que si se ha encontrado una cadena con k unos, se ha examinado aproximadamente 2^k elementos. Este es un buen punto de partida. Al tener una lista de elementos que se distribuyen uniformemente entre 0 y $2^k - 1$, se puede contar el número máximo del prefijo más grande de unos en representación binaria y esto dará una estimación razonable.

El problema es que la suposición de tener números distribuidos uniformemente de $\{0 \cdots 2^k - 1\}$ es demasiado difícil de lograr (los datos que encontramos no es en su mayoría números, casi nunca distribuidos de manera uniforme, y puede estar entre cualquier valor. Pero usando una buena función hash se puede suponer que los bits de salida se distribuirían uniformemente y la mayoría de las funciones hashing tienen salidas entre 0 y $2^k - 1$. Entonces, lo que hemos logrado hasta ahora es que podemos estimar el número de elementos únicos con la cardinalidad máxima de k bits almacenando solo un número de bits de registro de tamaño $\log(k)$.

2.4.3. *LogLog* y *Super-LogLog*

Los sketches LogLog (LL), y Super-LogLog (SLL) [66], se utilizan para estimar el número de elementos distintos en un conjunto, empleando solo un pequeño espacio de memoria auxiliar y una sola pasada sobre cada elemento. El Super-LogLog Sketch es una versión mejorada del LogLog Sketch básico. En los siguientes párrafos se describe tanto el algoritmo básico de LogLog como las técnicas a través de las cuales se logran las mejoras. Cuando no se dice nada, el Super LogLog funciona de la misma manera que el algoritmo LogLog básico.

La estructura de datos es una matriz de m unidades de memoria que toman solo $[\log_2(\log_2(N_{max}))]$ bits cada uno, donde N_{max} es el número máximo de elementos distintos esperados. Todas las posiciones de la matriz se inicializan a cero. El valor de m determina la precisión del algoritmo.

Al igual que en el PCSA Sketch, se necesita una función hash h para transformar los elementos de entrada en cadenas binarias de tamaño H. El valor H, correspondiente al largo de los elementos hasheados, debe satisfacer $H \geq \log_2 m + \left[log_2\left(\frac{N_{max}}{m}\right) + 3\right]$. Una segunda función, f, es necesaria para encontrar el rango del primer bit "1", contando desde la izquierda, en una secuencia de bits. Por lo tanto $f(1\dots) = 1$, $f(001\dots) = 3$, $f(0^k) = k + 1$, etc.

En el algoritmo Super-LogLog, es posible reducir el tamaño de cada unidad de memoria a $\left[\log_2\left[log_2\left(\frac{N_{max}}{m}\right) + 3\right]\right]$ bits. El algoritmo soporta dos tipos de operaciones, una para agregar un nuevo elemento a la estructura de datos y otra para estimar el número de elementos distintos agregados.

Durante el procedimiento de actualización, cada vez que llega un elemento i, este es inmediatamente mapeada por la función de hash h. Siendo $k = \log_2 m$. El valor de los primeros k bits de $h(i)$ es un índice j a una posición en un arreglo de tamanõ m. La posición j en el arreglo contiene un valor $M(j)$, que luego se establece al máximo entre su valor anterior y la salida de la función f aplicada a la representación binaria de $h(i)$ sin sus primeros k bits.

Durante el procedimiento de estimación, el valor E devuelto por el algoritmo LogLog básico, correspondiente al número estimado de elementos distintos agregados a la estructura de datos, viene dado por la ecuación (2.1).

$$E = \alpha_m \cdot m \cdot 2^{\frac{1}{m}\sum_j M(j)} \tag{2.1}$$

El error estándar mide, en proporción al número real de elementos distintos, la desviación que se espera en el resultado estimado. Una aproximación de este valor, utilizando el algoritmo básico de LogLog, viene dada por la ecuación (2.2).

$$\sigma \approx \frac{1{,}30}{\sqrt{m}} \tag{2.2}$$

donde σ representa el error estándar.
En el algoritmo Super-LogLog, solo una parte de la matriz se utiliza para calcular el número de elementos distintos. Esta parte corresponde a los $m_0 = \theta_0 m$ valores más pequeños almacenados en la matriz. La constante θ_0 es un número real entre 0 y 1, que produce resultados casi óptimos cuando su valor es 0.7. El valor devuelto por el Super-LogLog viene dado por la ecuación (2.3).

$$E = \alpha_{m_0} \cdot m_0 \cdot 2^{\frac{1}{m_0}\sum^* M(j)} \tag{2.3}$$

donde $\sum^{*} M^{(j)}$ indica la suma de los valores en las posiciones seleccionadas de la matriz.

Usando las mejoras del algoritmo Super-LogLog, la precisión aumenta. El error estándar σ ahora viene dado por la ecuación (2.4).

$$\sigma \approx \frac{1{,}05}{\sqrt{m}} \tag{2.4}$$

2.4.4. *HyperLogLog*

El HyperLogLog (HLL) [67], es una mejora sobre los Sketches LogLog y Super-LogLog. El algoritmo fue desarrollado para estimar el número distinto de elementos de un conjunto, mientras que es más eficiente en memoria que sus predecesores. Su estructura de datos es la misma que sus versiones anteriores: una matriz de m buckets con $[\log_2(\log_2(N_{max}))]$ bits cada uno. Todas las m unidades de memoria están inicializadas con cero. Tiene una función hash h, que mapea los elementos de entrada en valores hash cuyos bits son independientes y cada uno tiene una probabilidad de ocurrencia de 0.5. Otra función f, permite que sea posible encontrar el bit "1" más a la izquierda en una cadena binaria. El algoritmo soporta los mismos dos tipos de operaciones.

En la figura 2.2 se presenta la estructura y describe el funcionamiento HLL. Este es caracterizado como un arreglo A de 2^p celdas. El parámetro p que es definido por el usuario y está relacionado con la precisión de la estimación de cardinalidad, determina el tamaño de la estructura de datos.

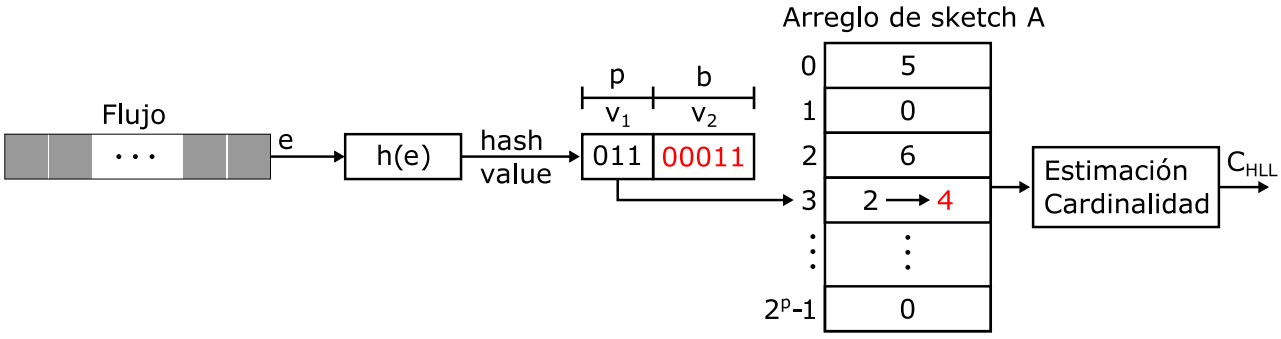

Figura 2.2: Sketch genérico para estimación de elementos distintos.

La Fig. 2.2, muestra la actualización de una celda del arreglo del sketch A. Donde $v_1 = 011_2 = 3$ indica la posición de la celda y v_2 se ocupa para calcular el 1 más a la izquierda, que en este caso está en la posición 4. Antes de la actualización, $A[3] = 2$, y porque $4 > 2$, el algoritmo actualiza el valor almacenado en la celda $A[3] = 4$. Es importante retener que el valor máximo

almacenado en cualquier celda en A es el número de ceros a la izquierda en v_2 más uno. Por lo tanto, la cantidad de bits necesarios para cada celda en A es $\lfloor \log(b) \rfloor + 1$.

HLL estima su cardinalidad calculando primero la media armónica Z de todos los valores en A como se muestra en la ecuación (2.5):

$$Z = \sum_{i=0}^{|A|-1} 2^{-A[i]}. \tag{2.5}$$

En seguida, el algoritmo estima la cardinalidad C_{HLL} del flujo usando ecuación (2.6):

$$C_{HLL} = \alpha_A \frac{|A|^2}{Z}, \tag{2.6}$$

donde α_A es un parámetro de corrección de sesgo que depende del número de celdas en la matriz A [67].

La complejidad espacial del sketch HLL es $\mathcal{O}(|A| \log \log N)$ y logra un error estándar de $1{,}03/\sqrt{|A|}$. Por lo tanto, aumentar el número de celdas en A disminuye el error de estimación. Sin embargo, para flujos con cardinalidad baja en comparación con $|A|$, la mayoría de las celdas en A son cero y el error aumenta. Heule et al. [68] muestran que, cuando $C_{HLL} \leq 2{,}5|A|$, la cardinalidad se puede estimar mejor usando ecuación (2.7), donde n_z es el número de ceros en A, como se muestra en ecuación (2.7):

$$C_{HLL} = |R| \log \frac{|R|}{n_z}. \tag{2.7}$$

2.4.5. Discusión

En esta sección se han presentado cuatro tipos de algoritmos de conteo de elementos distintos: PCSA, LL, SLL y HLL. Estos algoritmos de conteo probabilístico para aplicaciones de bases de datos de Flajolet-Martin en [65], con mejoras adicionales en los documentos LogLog contando grandes cardinalidades por Durand-Flajolet en [66], y HyperLogLog : el análisis de un algoritmo de estimación de cardinalidad casi óptimo por Flajolet et al. en [67].

En [65], Flajolet y Martin señalan que, dada una buena función hash, podemos tomar cualquier conjunto arbitrario de datos y convertirlo en uno del tipo que necesitamos, con valores dis-

tribuidos uniformemente (pseudo-aleatorios). Además, observaron que hay otros patrones que podemos usar para estimar el número de valores únicos, y algunos de ellos funcionan mejor que registrar el valor mínimo de los elementos hash. La métrica usada por estos autores cuenta el número de bits '0', al comienzo de los valores hash (*leading* ceros). Se ha visto que en datos aleatorios, una secuencia de cero k bits ocurrirá una vez en cada 2^k elementos, en promedio. Así que, todo lo que se necesita hacer es buscar estas secuencias y registrar la longitud de la secuencia más larga para estimar el número total de elementos únicos. Un problema con el algoritmo de Flajolet y Martin en el formulario anterior es que los resultados varían significativamente. Una solución común era ejecutar el algoritmo varias veces con diferentes funciones hash y combinar los resultados de las diferentes ejecuciones, o sea, promediarlas, para obtener una estimación con buenos resultados estadísticamente, pero el hashing es costoso.

La solución introducida en [65], denominada promedio estocástico, consiste en simular el efecto de m experimentos con una sola función hash. En términos generales, se divide el flujo de entrada $h(M)$ en m substreams, que corresponde a una partición del intervalo unitario de valores hash en $[0, \frac{1}{m}[\frac{1}{m}, \frac{2}{m}[, \ldots, [\frac{m-1}{m}, 1]$. Entonces, uno mantiene los m $(O_1 \ldots O_m)$ observables correspondiente a cada una de las m substreams. Se espera que un promedio adecuado de $\{O_j\}$ produzca una estimación de cardinalidades con una calidad mejorada, debido a los efectos promedio, en proporción a $\frac{1}{\sqrt{m}}$, a medida que m aumenta. El beneficio de este enfoque es que requiere solo un número constante de operaciones elementales por elemento del multiset M (en oposición a una cantidad proporcional a m), mientras que ahora solo se necesita una función hash.

En el paper LogLog en [66], Durand y Flajolet utilizaron un enfoque ligeramente diferente (probablemente porque el hashing es algo costoso). En lugar de usar múltiples funciones hash, han usado solo una función hash, pero lo dividieron en dos partes. Uno se llama *buckets* (el número total de buckets es de 2^k), y otro, es básicamente el mismo que el valor hash. Al tener más buckets, disminuye la variación (usa un poco más de espacio, pero aún es pequeño). Usando habilidades matemáticas, pudieron cuantificar el error, que es dada por la ecuación (2.2). Donde m es el número de buckets. HyperLogLog mejora LogLog de dos formas principales. Primero, usa una media armónica cuando combina los m buckets. Esto reduce el impacto de recuentos inusualmente altos. En segundo lugar, realiza correcciones para dos casos extremos: el extremo pequeño, cuando no están ocupados los m buckets, y el extremo grande, cuando las colisiones de hash causan subestimaciones. Estas correcciones se logran usando ideas de conteo lineal. La principal contribución de Flajolet et al en [67], fue de usar un tipo diferente de promedio, tomando la media armónica en lugar de la media geométrica. Al hacer esto, se pudo reducir el error a $1{,}04/\sqrt{m}$.

2.5. Entropía empírica

Según [36], la entropía empírica se refiere a la entropía de información calculada a partir de la distribución empírica de un conjunto de datos. Es una función de agregación ampliamente utilizada para el descubrimiento del conocimiento, así como la base de otras funciones de agregación, como la información mutua. Por ejemplo, la información mutua puede usarse para medir la relevancia de una característica con respecto a una columna objetivo para la predicción [69]. También se puede usar para evaluar la ganancia de información al agregar una función como criterio de división de datos para el aprendizaje del árbol de decisión [70]. Además, la información mutua condicional se utiliza para medir la redundancia entre características [71].

2.5.1. Definición matemática de la entropía

La definición matemática de la entropía empírica de un conjunto de datos viene dada por la ecuación (2.8).

$$H = -\sum_{i=1}^{N} \frac{m_i}{M} \log_2 \frac{m_i}{M}. \tag{2.8}$$

Donde, se asume que todos los elementos que llegan a través de la secuencia se extraen del conjunto $[N] = \{1, 2, 3, \cdots, N\}$. Denotaremos la frecuencia del elemento $i \in [N]$, (por ejemplo, el número de paquetes vistos en el puerto i), por m_i, y el número total de elementos en la secuencia por: $M = \sum_{i=1}^{N} m_i$. El enésimo elemento observado en la secuencia será denotado por $aj \in [N]$. El valor de la entropía en la ecuación (2.8) depende de N, lo que dificulta la comparación de la entropía de conjuntos de datos de diferentes tamaños. En esos casos, es más conveniente utilizar la entropía normalizada, que tiene un valor entre $0 - 1$ y se define como la ecuación (2.9).

$$H_{norm} = \frac{H}{\log_2 N}, \tag{2.9}$$

donde, N es la cantidad de elementos distintos que aparecen en la secuencia. Esto es conveniente porque permite normalizar la entropía en un rango entre $0 - 1$, y simplifica el análisis de la variación de la entropía observada en el flujo de tiempo.

Capítulo 3. Metodología

En esta sección, primero se define matemáticamente la entropía empírica y normalizada. Posteriormente, se discute la dificultad del cálculo real de la entropía para grandes flujos de red. Más adelante, se exponen los dos enfoques propuestos para estimar la entropía de forma eficiente. Para cada uno de ellos, se describen las definiciones matemáticas, los algoritmos implementados y las estructuras de datos usadas.

3.1. Restricciones en el cálculo real de la entropía

El cálculo de la entropía empírica exacta en un gran conjunto de datos puede ser costoso, debido a que el conjunto múltiple a tratar es demasiado grande para mantenerse en la memoria. Como se ve en la ecuación (2.8), su definición requiere calcular las frecuencias m_i de cada elemento, además de los valores de M y N. Podemos calcular el valor de M simplemente contando los elementos a medida que llegan. Sin embargo, calcular el número de elementos distintos N y las frecuencias m_i de cada uno de los elementos de la secuencia no es sencillo. En las aplicaciones de análisis de redes en tiempo real, el cálculo exacto de estos valores puede ser difícil, ya que requiere crear y mantener un contador para cada elemento diferente visto en esta secuencia, y actualizar estos contadores a la misma velocidad a la que se reciben los paquetes en el enlace de datos. Una solución práctica es, relajar la restricción de un cálculo exacto. De esta forma, se puede calcular un aproximación de la entropía empírica de manera eficiente, debido a que, en muchas aplicaciones prácticas es aceptable una tolerancia de unos pocos puntos porcentuales sobre el resultado.

3.2. Enfoques propuestos para estimar la entropía

Dado las restricciones mencionadas en la sección 3.1, para el cálculo real de la entropía empírica para grandes flujos de red, se presentan dos enfoques que proporcionan un método para aproximar el valor de la entropía empírica de un gran conjunto de datos en procesamiento en tiempo real, utilizando un espacio de memoria sub-lineal. Para estimar estos valores con gran precisión y rendimiento, se utilizan estructuras de datos basadas en sketches que requieren un espacio sub-lineal y permiten explotar un procesamiento paralelo de grano fino.

3.2.1. Enfoque 1

Este enfoque consiste en aproximar el cálculo de la entropía de un gran conjunto de datos, considerando los K elementos más frecuentes (o sea, los elementos top-K).

La definición matemática de la entropía empírica considerando este enfoque, que se obtiene modificando la ecuación (2.8), es dada por la ecuación (3.1), y su entropía normalizada es dada por la ecuación (3.2).

$$\hat{H} = -\sum_{i=1}^{K} \frac{m_i}{L} \log_2 \frac{m_i}{L}. \tag{3.1}$$

Donde, K es el número de las direcciones IP distintas más frecuentes, m_i indica la frecuencia de la dirección IP i (o sea, la cantidad de paquetes generados por la dirección de IP i) en el flujo de datos y $L = \sum_{i=1}^{K} m_i$ es el número de ocurrencias de los elementos top-K (o sea, la cantidad total de paquetes generados por los elementos top-K).

$$\hat{H}_{norm} = \frac{\hat{H}}{\log_2 K}, \tag{3.2}$$

donde, definimos que K es la cantidad de los elementos considerados heavy hitters que aparecen en la secuencia.

La ecuación (3.1) muestra que el método de estimación requiere seleccionar el número de los K elementos más frecuentes, y calcular tanto el recuento de frecuencia m_i para cada K elemento más frecuente, como el número total de ocurrencias de los elementos top-K, definido por $L = \sum_{i=1}^{K} m_i$. De esta forma, para este enfoque se usa un sketch para el estimar la frecuencia m_i de los elementos que llegan en la secuencia y una cola de prioridad (PQ) para mantener de forma ordenada los elementos top-K y sus respectivas frecuencias m_i.

3.2.2. Enfoque 2

Este enfoque (que es, el principal), consiste en aproximar el cálculo de la entropía de un gran conjunto de datos, no solo tomando en cuenta los elementos top-K, más también, la contribución

de los elementos menos frecuentes (o sea, elementos de la cola), asumiendo que estos siguen una distribución uniforme simples. La idea es de calcular la entropía empírica empírica en dos partes. Tomando apenas aquellas direcciones de IP más frecuentes (o sea, heavy hitters), más de los elementos menos frecuentes. Para la parte correspondiente a los elementos menos frecuentes, basta que se conozca la cantidad total de paquetes generados, más la cantidad de paquetes generados por los elementos heavy hitters y asumir que estos siguen una distribución uniforme (o sea, tienen la misma probabilidad de ocurrencia).

La definición matemática de la entropía empírica considerando este enfoque, se obtiene separando la ecuación (2.8), de estimación de entropía empírica en dos partes. Una corresponde a los elementos más frecuentes y la otra a los elementos menos frecuentes, es dado por la ecuación (3.3):

$$H = -\left(\sum_{i=1}^{K} \frac{m_i}{M} \log_2 \frac{m_i}{M} + \sum_{i=K+1}^{N} \frac{m_i}{M} \log_2 \frac{m_i}{M} \right). \tag{3.3}$$

Suponiendo una elección de Kque captura elementos con las frecuencias más altas, el segundo componente contendrá los elementos con las frecuencias más bajas. Además, la varianza de estas frecuencias es pequeña en comparación con la de los K elementos más frecuentes. Simplificamos el cálculo de m_i en el segundo componente suponiendo que los elementos de este conjunto están distribuidos uniformemente, por lo que m_i es un valor constante para todos $i \in [K-1, N]$.

Definiendo $L = \sum_{i=1}^{K} m_i$ como el número total de ocurrencias de los elementos top-K, el número total de ocurrencias de los elementos en el segundo componente es $\sum_{i=K+1}^{N} m_i = M - L$. Además, el número de elementos distintos en el segundo componente es, $N - K$. Por lo tanto, suponiendo una distribución uniforme, podemos estimar $m_i = \frac{M-L}{N-K}$ for $K < i \leq N$. Sustituyendo estos valores en la ecuación (3.3), obtenemos la estimación de la entropía dada en la ecuación (3.4):

$$\hat{H} = -\left[\sum_{i=1}^{K} \frac{m_i}{M} \log_2 \frac{m_i}{M} + \sum_{i=K+1}^{N} \frac{\left(\frac{M-L}{N-K}\right)}{M} \log_2 \frac{\left(\frac{M-L}{N-K}\right)}{M} \right]. \tag{3.4}$$

Dado que $\frac{M-L}{N-K}$ es una constante, simplificamos la ecuación (3.4) para obtener la ecuación (3.5),

que es la estimación de entropía empírica:

$$\hat{H} = -\Big[\sum_{i=1}^{K} \frac{m_i}{M} \log_2 \frac{m_i}{M} + \frac{M-L}{M} \log_2 \left(\frac{M-L}{M(N-K)}\right)\Big]. \tag{3.5}$$

De manera análoga a la entropía normalizada exacta, definimos la entropía estimada normalizada en la ecuación (3.6):

$$\hat{H}_{norm} = \frac{\hat{H}}{\log_2 N}. \tag{3.6}$$

La ecuación (3.5) muestra que el método de estimación requiere seleccionar el número de los K elementos más frecuencia, y calcular tanto el recuento de frecuencia m_i para cada K elemento más frecuente como el número total de elementos distintos en la secuencia de entrada N.

3.3. Algoritmos

En esta sección, se describen los algoritmos usados para estimar la entropía empírica según los dos enfoques mencionados en la sección 3.2.1 y 3.2.2, respectivamente. Estos utilizan estructuras de datos basadas en sketches que requieren un espacio sub-lineal y permiten el diseño y la implementación de un acelerador de hardware que opera con bajo uso de memoria y alto rendimiento.

3.3.1. Enfoque 1

El algoritmo 1 presenta las etapas para estimar la entropía empírica normalizada de un flujo de red según el enfoque 1.

El algoritmo 1 utiliza un *FreqSketch* para mantener una estimación de la frecuencia de todos los elementos en la secuencia, y una cola de prioridad PQ para almacenar los K elementos más frecuentes y sus respectivas frecuencias. Se estima la entropía normalizada del flujo calculando la entropía de los elementos almacenados en el PQ. La actualización de la PQ, funciona de la siguiente manera: cuando llega un elemento nuevo, este e se inserta en PQ si la cola no está llena. Si la PQ está llena, e remplaza al elemento menos frecuente en la PQ solo si su

Algoritmo 1 Estimación de entropía empírica normalizada según el enfoque 1

Entrada: *flujo de red*
Salida: *entropía normalizada* $\hat{H}_{norm}$
$L \leftarrow 0$, $K \leftarrow 0$, $\hat{H} \leftarrow 0$
foreach elemento e **in** *flujo de red*
    $FreqSketch.update(e)$
    $PQ.update(e, FreqSketch.estimate[e])$
end
foreach frecuencia m_i **in** PQ
    $L \leftarrow L + m_i$
    $K \leftarrow K + 1$
end
foreach frecuencia m_i **in** PQ
    $\hat{H} \leftarrow \hat{H} + \frac{m_i}{L} \log_2 \frac{m_i}{L}$
end
$\hat{H}_{norm} \leftarrow \frac{\hat{H}}{\log_2 K}$

frecuencia estimada es mayor. Si el elemento e ya existe en la PQ, se actualiza su frecuencia. También importa referir, que el uso del *FreqSketch* en conjunto con la PQ torna el algoritmo más eficiente. Si solo se usase este último, para almacenar y actualizar las frecuencias, no sería posible mantener la frecuencia de aquellos elementos que han sido descartados de la PQ. Visto que la estrutura *FreqSketch* permite almacenar y actualizar de manera eficiente a la frecuencia de los elementos descartados de la PQ, o que aún no se han insertado porque su frecuencia es menor que la elemento menos frecuente en el PQ.

En la primera etapa, el algoritmo procesa los elementos e del flujo en una pasada para estimar la ocurrencia de todos los elementos de la secuencia. Después de la estimación de frecuencia m_i, a través del *FreqSketch*, se utiliza el elemento e y su frecuencia m_i estimada para actualizar el contenido de la cola de prioridad PQ. La cola de prioridad PQ es usada para almacenar los K elementos más frecuentes y sus frecuencias m_i.

En la segunda etapa, el algoritmo hace una pasada sobre la cola de prioridad PQ que contiene los elementos top-K y sus respectivas frecuencias m_i. En este proceso, se cuentan la cantidad de los K elementos más frecuentes (que es la estimación de la cantidad de elementos distintos), y se suman las frecuencias de cada uno de los K elementos para estimar la cantidad total de paquetes generados por las direcciones heavy hitters $L = \sum_{i=1}^{K} m_i$.

En la tercera etapa, el algoritmo hace una segunda pasada sobre la cola de prioridad PQ y calcula la entropía estimada $\hat{H}$ según la ecuación (3.1). Posteriormente, el algoritmo calcula la

entropía normalizada, según la ecuación (3.2).

3.3.2. Enfoque 2

El algoritmo 2 presenta las etapas para estimar la entropía empírica normalizada de un flujo de red según el enfoque 2.

Algoritmo 2 Estimación de entropía empírica normalizada según el enfoque principal

Entrada: *flujo de red, tamaño máximo de la cola de prioridad* K_{pq}*, cotas* d *y* w *del sketch de frecuencia, precisión* p *del sketch de cardinalidad, función hash* h
Salida: *entropía normalizada* $\hat{H}_{norm}$
$L \leftarrow 0$, $K \leftarrow 0$, $M \leftarrow 0$, $\hat{H} \leftarrow 0$
$FreqSketch.initialize(d, w, h)$
$PQ.initialize(K_{pq})$
$CardSketch.initialize(p, h)$
foreach elemento e **in** *flujo de red*
    $CardSketch.update(e)$
    $FreqSketch.update(e)$
    $PQ.update(e, FreqSketch.estimate[e])$
    $M \leftarrow M + 1$,
end
foreach frecuencia m_i **in** PQ
    $L \leftarrow L + m_i$
    $K \leftarrow K + 1$
    $\hat{H} \leftarrow \hat{H} + \frac{m_i}{M} \log \frac{m_i}{M}$
end
$N \leftarrow CardSketch.estimate()$
$\hat{H} \leftarrow \hat{H} + \frac{M-L}{M} \log_2 \frac{M-L}{M(N-K)}$
$\hat{H}_{norm} \leftarrow \frac{\hat{H}}{\log_2 N}$

El algoritmo 2 utiliza un contador real M para mantener la cuenta total de los elementos vistos en el flujo, una estructura de datos *FreqSketch* para mantener una estimación de la frecuencia de todos los elementos en la secuencia, una estructura de datos denominada *CardSketch* para estimar la cardinalidad de los elementos del flujo, y una cola de prioridad PQ para almacenar los K elementos más frecuentes y sus frecuencias m_i. Se estima la entropía normalizada del flujo calculando la suma de la entropía de los elementos almacenados en el PQ más la entropía generada por los elementos menos frecuentes asumiendo el presupuesto mencionado en la sección 3.2.2.

En la primera etapa, el algoritmo procesa los elementos e del flujo de red en una pasada para estimar la cantidad total de elementos del flujo $M = \sum_{i=1}^{N} m_i$, actualiza el estado del sketch $FreqSketch$ que proporciona la frecuencia m_i de todos los elementos del flujo, sketch $CardSketch$ que proporciona la cardinalidad N. Este también actualiza la cola de prioridad PQ, usando la estimación actual de e proporcionada por la estructura $FreqSketch$. El funcionamiento de la cola es similar al de la PQ presentada en el algoritmo 1. El tamaño máximo de PQ es un parámetro proporcionado por el usuario K_{pq}.

En la segunda etapa, el algoritmo hace una pasada sobre la cola de prioridad PQ que contiene los K elementos más frecuentes del flujo de red y sus frecuencias m_i estimadas por el $FreqSketch$. Este calcula la cantidad de los elementos más frecuentes K contando los elementos de la PQ, y la cantidad total de paquetes generados por estos elementos top-K $L = \sum_{i=1}^{K} m_i$. También calcula la entropía $\sum_{i=1}^{K} \frac{m_i}{M} \log_2 \frac{m_i}{M}$, generada por el primer término de ecuación (3.5).

En la tercera etapa, el algoritmo calcula el segundo término de la ecuación (3.5) usando el valor estimado de N proporcionado por *CardSketch*, y devuelve la estimación de entropía empírica normalizada $\hat{H}_{norm}$.

3.3.3. Algoritmos para estimación de ocurrencias

Los algoritmos 3-5 presentan las operaciones de inicialización, actualización y estimación soportadas por el sketch CM-CU.

Algoritmo 3 Inicialización: CM-CU

1: **entrada:** dimensiones d y w, función hash $h_j, j \in [1..d]$
2: $C[i][j] \leftarrow 0, i \in [1..d], j \in [1..w]$

El algoritmo 3 presenta el proceso de inicialización del sketch, en donde se pone a cero las celdas del contador de la matriz C. En este proceso, los parámetros del sketch d y w, son definidos por el usuario.

El algoritmo 4 presenta el proceso de actualización del sketch, en donde los elementos de entrada e que llegan en la secuencia, son mapeados por las d funciones de hash h_j y se calcula el valor mínimo de los contadores seleccionados, y después actualizar/incrementar el contador mínimo.

El algoritmo 5 presenta el proceso de estimación del sketch, en donde con las d funciones de hash se mapea el elemento e, y se obtiene el mínimo del contador como la estimación deseada.

Algoritmo 4 Actualización: CM-CU

entrada: elemento del flujo e
for $j = 1$ to d **do**
  $min\_est \leftarrow \min\{C[j, h_j(\mathrm{e})]\}, j \in [1, d]$
  if $C[j, h_j(\mathrm{e})] = min\_est$ **then**
    $C[j, h_j(\mathrm{e})] \leftarrow C[j, h_j(\mathrm{e})] + 1$
  end if
end for

Algoritmo 5 Estimación: CM-CU

entrada: elemento del flujo e
return $\min\{C[j, h_j(e)]\}, j \in [1, d]$

3.3.4. Algoritmos para estimación de cardinalidad

Los algoritmos 6-8 describen los algoritmos de inicialización, actualización y estimación del algoritmo HLL.

Algoritmo 6 Inicialización: HyperLogLog

entrada: parámetro de precisión p, función hash h
Sea $|A| = 2^p$
$\alpha_A = 0{,}7213/(1 + 1{,}079/|A|)$ **considerando** $|A| \geq 128$.
$A[i] \leftarrow 0, i \in [0, |A| - 1]$

El algoritmo 6 presenta el proceso de inicialización del HLL sketch, que resulta con que todos los elementos de A estén cero y también se define el valor del corrección de sesgo α_A cuando $|A| \geq 128$. Para valores más pequeños de $|A|$, α_A es una constante predefinida [67].

Algoritmo 7 Actualización: HyperLogLog

entrada: elemento del flujo de red e.
$x \leftarrow h(e)$
$v_1 \leftarrow < x_{31}, \cdots, x_{32-p} >_2$
$v_2 \leftarrow < x_{31-p}, \cdots, x_0 >_2$
$A[v_1] \leftarrow \max\{A[v_1], ldz(v_2) + 1\}$

El algoritmo 7 demuestra el proceso de actualización del sketch, en donde se calcula el valor hash x del elemento de entrada e, y se usan los p bits más significativos de x para indicar la posición de la celda en A. En este proceso, también se usa la función *ldz()* para calcular la posición del 1 más a la izquierda en los b bits menos significativos de x contando el número de ceros a la izquierda y actualiza el contenido de la celda seleccionada en A.

Algoritmo 8 Estimación: HyperLogLog

1: $Z \leftarrow \sum_{i=0}^{|A|-1} 2^{-A[i]}$
2: $C_{HLL} \leftarrow \alpha_A \frac{|A|^2}{Z}$
3: **if** $C_{HLL} \leq 2{,}5|A|$ **then**
4: $\quad n_z \leftarrow ContadordeCeros(A)$
5: $\quad C_{HLL} \leftarrow 2{,}5 \log(|A|/n_z)$
6: **end if**
7: **return** C_{HLL}

El algoritmo 8 describe el proceso de estimación, en donde se calcula la media armónica del contenido de A y se estima la cardinalidad de la secuencia.

3.3.5. Función Hash

El algoritmo 9 presenta el proceso de generación la función MurmurHash3 de 32 bits. Murmur3 se usa ampliamente en algoritmos de transmisión porque es rápido de calcular y logra bajas tasas de colisión [72].

Algoritmo 9 MurmurHash3

1: **entrada:** elemento del flujo de red e, semilla s
2: **salida:** *valor hash*
3: $k_0 \leftarrow$ `0xcc9e2d51` $\times$ e
4: $k_1 \leftarrow (k_0 \ll 15)|(k_0 \gg 17)$
5: $k_2 \leftarrow$ `0x1b873593` $\times k_1$
6: $k_3 \leftarrow k_2 \oplus s$
7: $k_4 \leftarrow (k_3 \ll 13)|(k_3 \gg 19)$
8: $k_5 \leftarrow 5 \times k_4 +$ `0xe6546b64`
9: $k_6 \leftarrow k_5 \oplus 4$
10: $k_7 \leftarrow k_6 \oplus (k_6 \gg 16)$
11: $k_8 \leftarrow$ `0x85ebca6b` $\times k_7$
12: $k_9 \leftarrow k_8 \oplus (k_8 \gg 13)$
13: $k_{10} \leftarrow$ `0xc2b2ae35` $\times k_9$
14: $k_{11} \leftarrow k_{10} \oplus (k_{10} \gg 16)$
15: **return** k_{11}

El algoritmo 9, muestra que cada instancia de la función hash usa una semilla elegida al azar y calcula el valor hash usando una secuencia de multiplicaciones, sumas y operaciones lógicas bit a bit como or, xor y desplazamiento de bits .

Capítulo 4. Arquitectura del sistema

4.1. Arquitecturas de hardware

En esta sección, se describen las arquitecturas aceleradoras de hardware usadas para estimar la entropía empírica según los dos enfoques mencionados en la sección 3.2.1 y 3.2.2, respectivamente.

4.1.1. Arquitectura aceleradora de hardware del enfoque 1

En la Figura 4.1, se presenta la arquitectura del acelerador de hardware para estimar la entropía empírica según el enfoque propuesto en la sección 3.2.1. En conformidad con el algoritmo 1, la arquitectura aceleradora utiliza 3 bloques principales: *CM-CU sketch* para estimar la frecuencia m_i de los elementos e que llegan del flujo de red, *Cola de Prioridad* para almacenar el elemento e con su frecuencia y *Entropía* para estimar el valor $\hat{H}$ según la (3.1).

Figura 4.1: Arquitectura general del sistema según el enfoque de la sección 3.2.1.

4.1.2. Arquitectura aceleradora de hardware del enfoque 2

La Figura 4.2 muestra la arquitectura del acelerador de hardware para estimar la entropía empírica según el enfoque propuesto en la sección 3.2.2. En conformidad con el algoritmo 2, la arquitectura aceleradora utiliza 5 bloques principales: *CM-CU sketch* para estimar la frecuencia m_i de los elementos e que llegan del flujo de red, *Cola de Prioridad* para almacenar el elemento e con su frecuencia, *Contador* para calcular el número total de elementos M en el flujo, *HLL sketch* para estimar la cardinalidad N y *Entropía* para estimar el valor $\hat{H}$ según la ecuación (3.5).

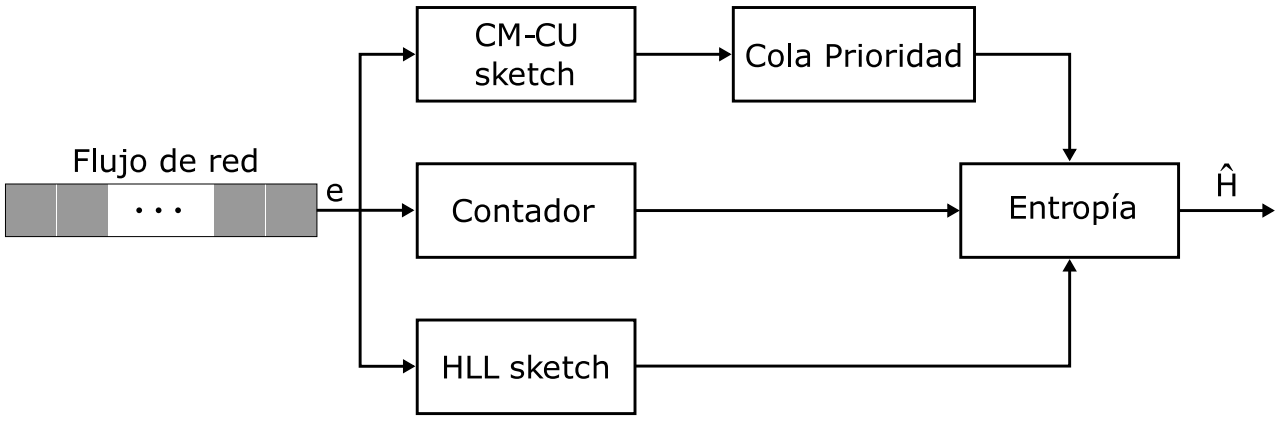

Figura 4.2: Arquitectura general del sistema según el enfoque de la sección 3.2.2.

4.1.3. Módulo Countmin-CU sketch

Para el enfoque 1 se implementa una arquitectura de la estructura CM-CU sketch para estimar la frecuencia de los elementos e del flujo de red, considerando los del parámetros $d = 4$ y $w = 16384$. La Figura 4.3 muestra la arquitectura del módulo CM-CU para este enfoque.

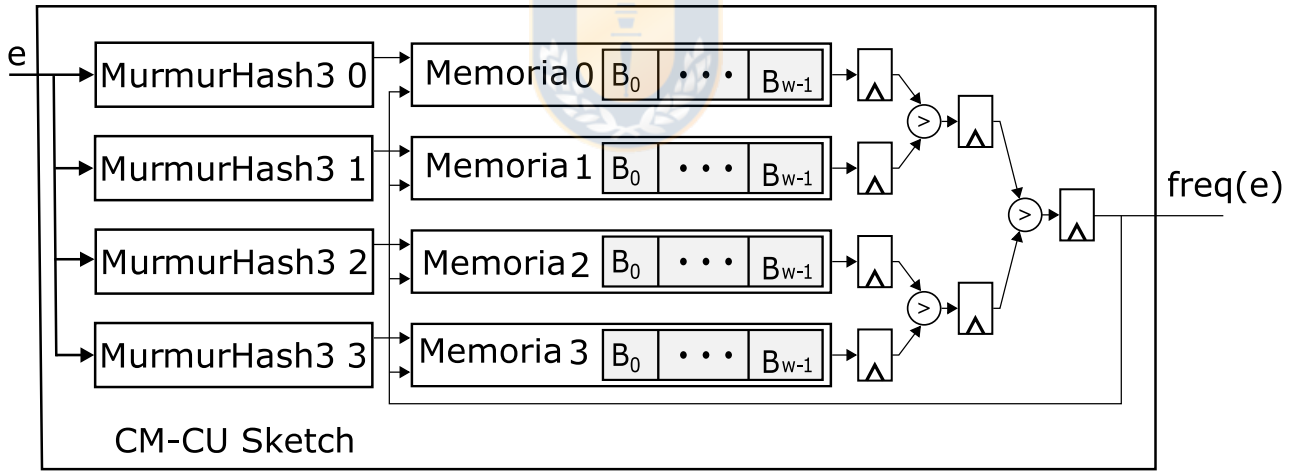

Figura 4.3: Arquitectura CM-CU sketch según el enfoque de la sección 3.2.1.[1]

El módulo de la Figura 4.3 se implementa usando d bloques de memoria de 16384 contadores cada uno. Se utiliza los 14 bits menos significativos (LSB) del valor de hash para acessar a cada uno de los contadores de los d bloques de memoria. Se implementa cada memoria de 16384 contadores usando 16 módulos de bloque ram's (BRAM) de 1024 elementos de 21 bits. Se utiliza una arbole de reducción de dos etapas para calcular el valor mínimo de los d contadores seleccionados por la función de hash. Este contador mínimo seleccionado, se incrementa en 1

durante la actualización del sketch.

A medida que el sistema recibe una entrada válida por ciclo y el sketch simplemente aumenta el contador inferior, la implementación tendrá un hazard de $\log_2(d)$ ciclos, lo que provocará una subestimación de la frecuencia cuando el valor hash entrante sea el mismo para $\log_2(d)+1$ ciclos. Para reducir el impacto del hazard incrementando todos los contadores en paralelo con la operación del árbol comparador y enviando el valor actualizado del contador a la salida del bloque de memoria cuando detectamos el *hazard*. Este enfoque nos permite reducir el hazard a $\log_2(d)-1$ ciclos.

Para el enfoque 2 se implementa una arquitectura de la estructura CM-CU sketch para estimar la frecuencia de los elementos e del flujo de red, considerando los del parámetros $d = 8$ y $w = 8192$. La Figura 4.4 muestra la arquitectura del módulo CM-CU para este enfoque.

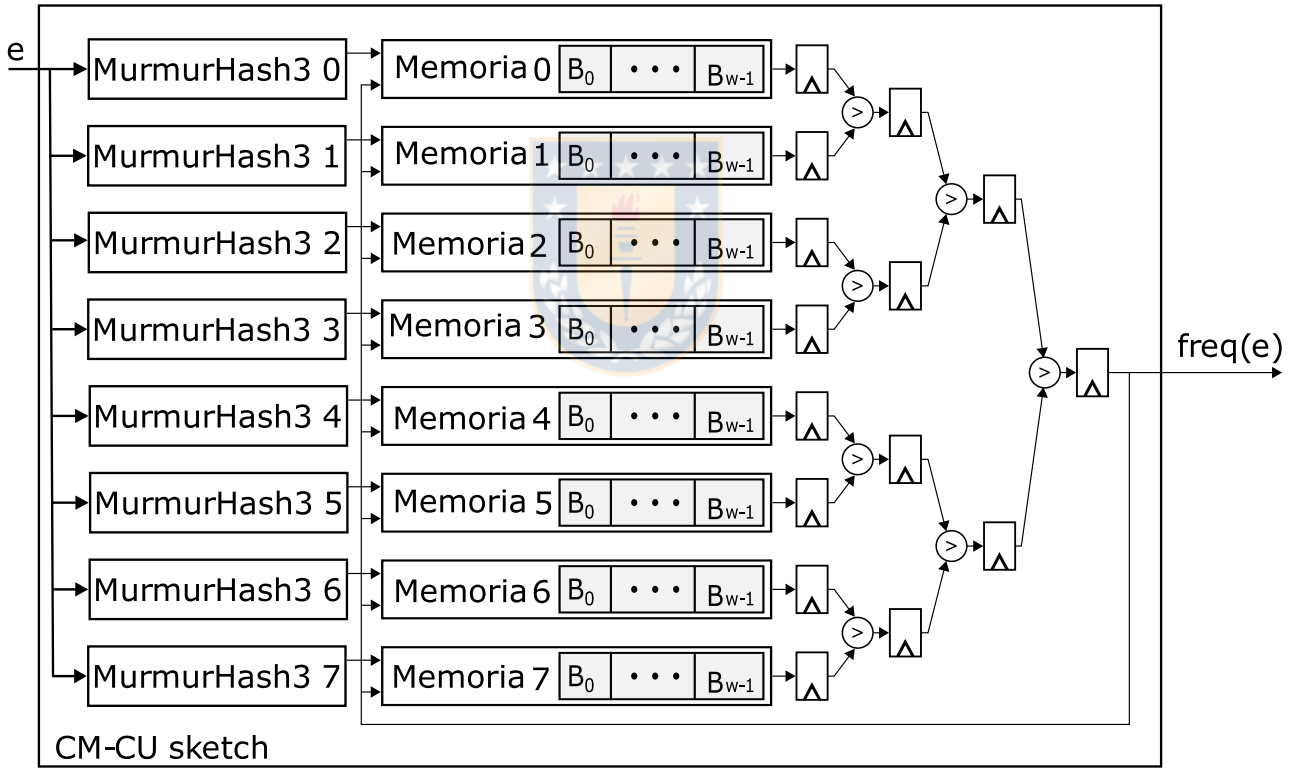

Figura 4.4: Arquitectura CM-CU sketch según el enfoque de la sección 3.2.2.[2]

El módulo de la Figura 4.4 se implementa usando d bloques de memoria de 8192 contadores cada uno. Se utiliza los 13 bits menos significativos (LSB) del valor de hash para acessar a cada uno de los contadores de los d bloques de memoria. Se implementa el módulo del sketch usando 32 BRAMs de 2048 elementos de 18 bits. Se utiliza una arbole de reducción de tres etapas para calcular el valor mínimo de los d contadores seleccionados por la función de hash. Este contador

mínimo seleccionado, por su vez se incrementa en 1 durante la actualización del sketch.

4.1.4. Módulo MurmurHash3

Este módulo computa la función de hash ocupando una implementación totalmente canalizada, basando-se en la versión de software de 32 bits descrita en [72]. El módulo recibe dos entradas, el valor para aplicar la función (campo del paquete) y una semilla, que es un número aleatorio y es diferente en cada instancia del módulo. Cada línea del Algoritmo 9 se ejecuta en un ciclo de reloj diferente de una manera totalmente canalizada y los valores k_i se almacenan en registros de canalización. El módulo tiene una latencia de 12 ciclos y un rendimiento de un hash válido por ciclo.

Para los dos enfoques el módulo MurmurHash3 implementa la función hash basada en el algoritmo 9. Este módulo recibe dos entradas y genera un resultado de salída de 32 bits, donde para el primer enfoque el CM-CU sketch usa 14 bits ($\log w$ bits) para direccionar las BRAM, y en el segundo enfoque se utilizan solo 13 bits, y en HLL sketch usa el valor de hash completo.

4.1.5. Módulo Cola de prioridad

Para almacenar los K elementos más frecuentes se utiliza un arreglo de colas de prioridad. El uso de una sola cola de prioridad requiere una implementación en memoria distribuida ya todos los elementos pueden cambiar en cada nueva entrada. Como la memoria distribuida es un recurso escaso, para los dos enfoques se propone una solución que consiste en utilizar múltiples colas de prioridad pequeñas y que comparten BRAM entre ellas. Cada elemento de la cola de prioridad es almacenada en diferentes BRAM, lo que permite tener acceso simultáneo a todos los elementos, y como el módulo recibe un dato válido por ciclo, solo una cola de prioridad estará en uso al mismo tiempo, lo que permitirá reutilizar las mismas BRAM en todas las colas de prioridad. La figura 4.5 muestra la arquitectura de hardware de la cola de prioridad. Este es un arreglo de S bancos de memoria de tamaño R con un bus de direcciones compartido, tal que $K_{pq} = R \times S$. Los bancos de memoria se implementan con recursos BRAM en una FPGA. El módulo usa esta arquitectura para implementar un arreglo de R colas de prioridades más pequeños (PQ_i) de S elementos cada uno. Para el enfoque 1, se consideró los siguientes parámetros de $R = 1024$ y $S = 10$, lo que resulta en el tamaño de la cola de $K_{pq} = 10240$, y para enfoque 2, se consideró los siguientes parámetros de $R = 1024$ y $S = 8$, lo que resulta el tamaño de la cola de $K_{pq} = 8192$.

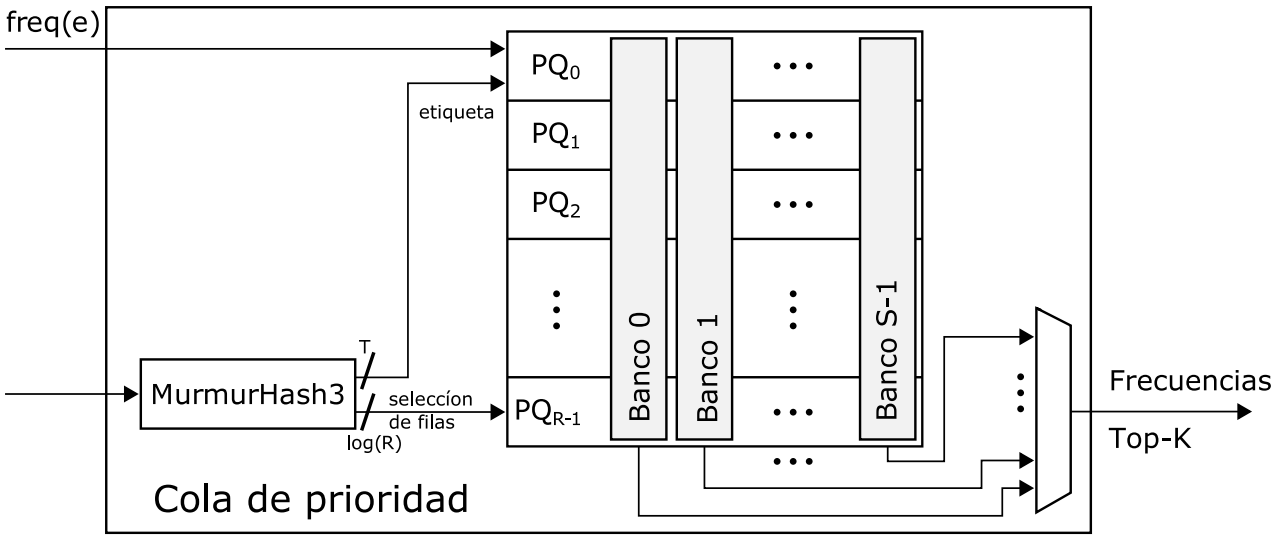

Figura 4.5: Arquitectura del módulo de cola de prioridad.[1]

Describiendo la arquitectura del módulo presentado en la Figura 4.5, se verifica que este módulo recibe dos entradas, el valor de la entrada e y su estimación de frecuencia obtenida a través del CM-CU sketch. Primeramente el módulo calcula el valor hash de una de sus entradas utilizando el módulo Murmurhash3 descrito en 4.1.4, donde por su vez este se reparte en dos valores T y $logR$, respectivamente. El módulo utiliza los $logR$ bits menos significativos del valor de hash para acceder/seleccionar la cola de prioridad correspondiente, y usa los T bits más significativos del valor de hash como una etiqueta para identificar el elemento e en la cola. La etiqueta se utiliza para determinar si el elemento de entrada e ya está en la cola, en donde se actualiza su valor de frecuencia.

Para el primer enfoque, se usan los 10 bits menos significativos del valor hash como una dirección para seleccionar un elemento de cada uno de las 10 blockRAM, y luego las etiquetas de los 10 elementos seleccionados se comparan con los 15 bits más significativos del valor hash para determinar si el elemento ys está presente en la cola. Simultáneamente, con recurso a 10 comparadores, se compara la frecuencia del elemento de entrada con las frecuencias de todos los elementos ya presente en la cola. Si el elemento no está presente, reemplaza el elemento con la frecuencia más baja en la cola, solo si su frecuencia es más alta. Si el elemento ya está en la cola, su frecuencia se actualiza, y el elemento se mueve por encima de los elementos con menor frecuencia. La salida de los comparadores de frecuencia devolverá uno para todos los elementos inferiores, y se mascara estos valores por la posición anterior del valor entrante. Se Implementa toda esta lógica para seleccionar datos de entrada y escribir señales de habilitación de BRAM, agregando una latencia de un ciclo adicional.

El final del proceso se determina mediante una señal de reinicio, en esta etapa el módulo vacía la cola un elemento a la vez, escribiendo ceros en la memoria y enviando el valor como una salida válida al módulo de cálculo de entropía. Este proceso tomará $1,024 \cdot n$ ciclos, con n como el número de elementos por cola. Para la implementación del segundo enfoque usa $K_{pq} = 8192$, con $R = 1024$ y $S = 8$. Por lo tanto, se usan 10 bits para direccionar el arreglo y 20 bits para la etiqueta. Los elementos en la PQ almacenan una etiqueta de 20 bits y un recuento de frecuencia de 16 bits, y los bancos de memoria se implementan con bloques BRAM de 36 bits en la FPGA.

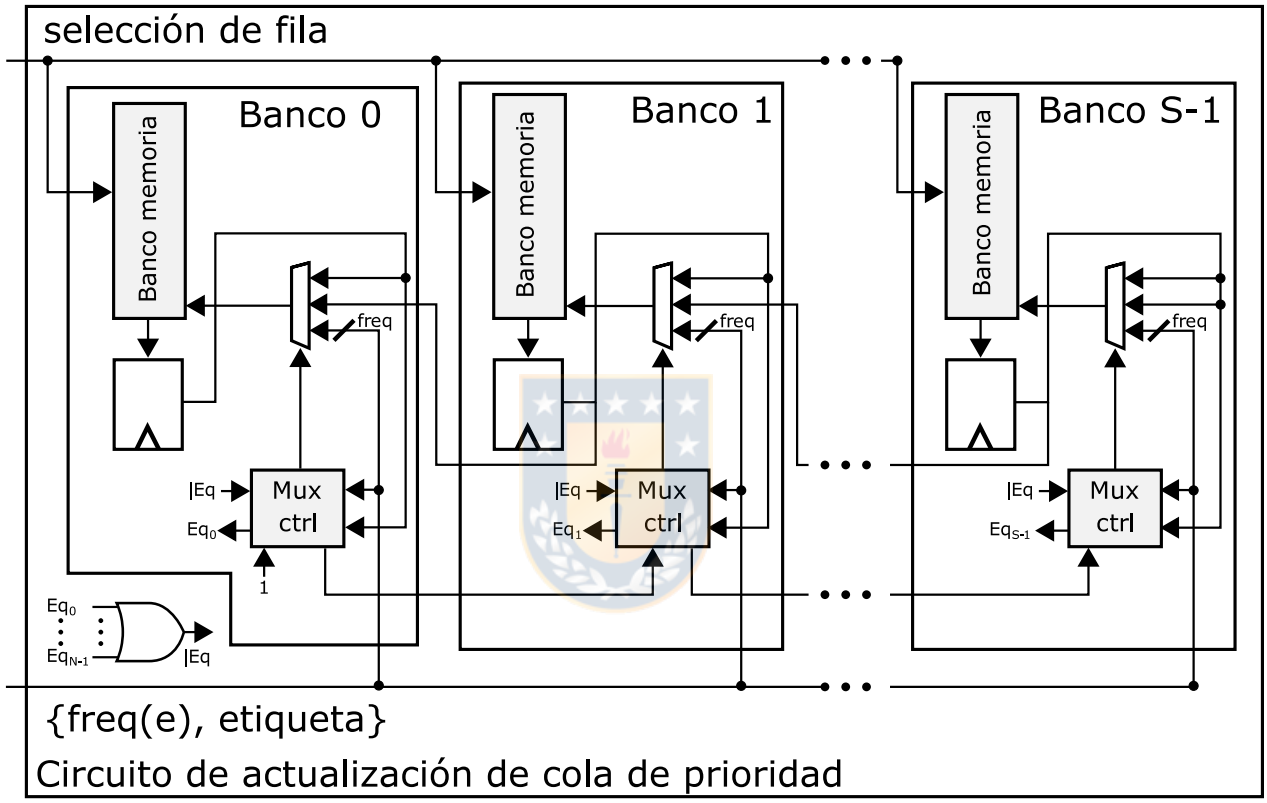

Figura 4.6: Circuito de actualización para la cola de prioridad.[2]

Para controlar las actualizaciones de la cola de prioridad se muestra la Figura 4.6. Donde para cada elemento i en la cola, el bloque *Mux ctrl*$_i$ determina si el elemento de entrada ya está almacenado en la posición correspondiente de la cola (Eq_i) y si la frecuencia de entrada es menor que el valor almacenado. Un OR lógico entre las señales Eq_i determina si el elemento de entrada ya está en la cola, en cuyo caso se actualizan su frecuencia y posición en la cola. Si el elemento no está en la cola, entonces se inserta en el lugar correcto a menos que su frecuencia sea menor que la del elemento de menor prioridad. Insertar un elemento o actualizar su posición en la cola requiere mover un subconjunto de los elementos en la cola una posición a la izquierda. Los bloques *Mux ctrl* determinan si se mantiene el valor actual, si se debe reemplazarlo con el

elemento de entrada o realizar un desplazamiento a la izquierda.

4.1.6. Módulo Hyperloglog

En la Figura 4.7, se presenta la arquitectura del módulo Hyperloglog, que se divide en dos bloques principales: actualización de sketch y estimación de cardinalidad.

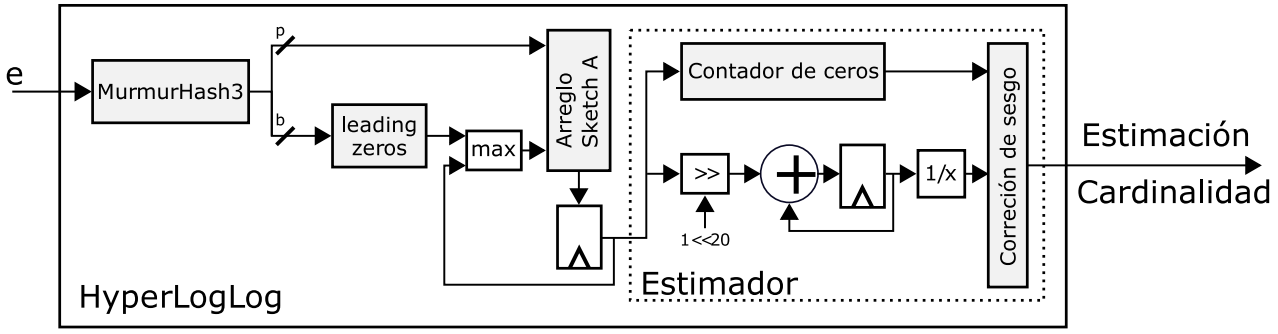

Figura 4.7: Arquitectura del módulo Hyperloglog.[2]

El módulo de la Figura 4.7 usa el módulo MurmurHash3 descrito en la sección 4.1.4 para calcular el valor de hash de 32 bits, que se reparte en dos valores p y b. El módulo utiliza los p bits más significativos para acceder al arreglo del sketch A (indica la posición actual del registro) y los b bits menos significativos para alimentar el bloque de leading ceros que calcula la posición del "1" más a la izquierda. La salida de este bloque se compara con el valor actual en el registro y el mayor valor se almacena nuevamente en el arreglo del sketch A.

El bloque Estimador funciona después de que la secuencia se haya procesado por completo. Este recibe los elementos del arreglo del sketch A, donde computa el número de ceros y calcula la línea 1 del algoritmo 8. Se utiliza aritmética de punto fijo con 20 bits fraccionarios para calcular el valor de z como descrito en el algoritmo 8. Luego en seguida computa el recíproco de Z usando un algoritmo de Newton-Raphson con 10 iteraciones y se multiplica el recíproco de Z por $\alpha_A|A|^2$ para calcular C_{HLL}. El último bloque del módulo Hyperloglog hace la corrección de sesgo en las líneas 3-5 del algoritmo 8 usando el número de ceros n_z en A calculado en el primer bloque.

En la figura 4.8 se presenta el bloque básico utilizado para calcular el "1" más a la izquierda en una palabra de n bits. El módulo Hyperloglog utiliza este bloque en tres situaciones: en el bloque ceros iniciales para la actualización de HLL, el divisor de Newton Raphson para obtener la primera estimación del recíproco y el cálculo de la función logarítmica.

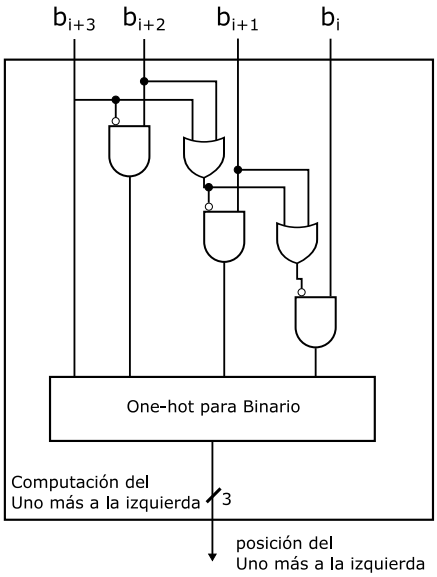

Figura 4.8: Bloque básico del "1" más significativo.[1]

4.1.7. Módulo de estimación de entropía

Para los dos enfoques, este módulo recibe el contenido de la para obtener los recuentos de frecuencia m_i para los K elementos más frecuentes del flujo, lo que toma $K_{pq} = 10240$ y $K_{pq} = 8192$ ciclos de reloj para los enfoques 1 y 2 respectivamente. Para el primer enfoque, se implementa un módulo final en el acelerador usa la información almacenada en la cola de prioridad (frecuencia de cada elemento y la frecuencia total), para estimar la entropía del tráfico de red usando la ecuación (3.1).

$$\hat{H} = \frac{1}{\log_2 K}\left(\log_2 L - \frac{1}{L}\sum_{i=1}^{K} m_i \cdot \log_2 m_i \right). \tag{4.1}$$

Para el segundo y principal enfoque, el módulo final en el acelerador no solo se utiliza los datos almacenados en la PQ, más también, utiliza el número total de paquetes de red almacenados en el contador de entrada M y la estimación de cardinalidad N del flujo del atráves del HLL sketch para estimar la entropía del tráfico de red usando la ecuación (4.2).

$$\hat{H} \;=\; -\frac{1}{M}\left[(M-L)\big(\log(M-L) - \log(M(N-K)\big) - L\log M + \sum_{i=1}^{K} m_i \log m_i \right]. \tag{4.2}$$

Capítulo 5. Resultados

En esta sección se presentan los resultados de los experimentos realizados para evaluar el método y el acelerador de hardware; bien como para determinar los parámetros del método para cada uno de los enfoques. También se describen las trazas de red utilizadas en los experimentos.

5.1. Enfoque 1

En esta sección se presentan los resultados obtenidos en los experimentos realizados para evaluar el método propuesto en el primer enfoque, que es dado por la ecuación (3.2). Inicialmente se describen las bases de datos ocupadas para realizar todos los experimentos, y luego se presentan las informaciones detalladas de esas trazas de red, tales como; nombre de identificación, cantidad total de paquetes, cantidad total de elementos distintos y el valor de la entropía exacta de todos elementos de la traza, calculada según la ecuación (2.9). En seguida se presentan los resultados de la evaluación de la entropía en función del valor de K. Luego con el valor de K definido, se presentan los resultados de la evaluación de la estimación de la entropía considerando distintos sketches de conteo de frecuencia de distintas dimensiones. Posteriormente con el sketch definido para la implementación, se evalúa la incorporación del arreglo de colas. A continuación se evalúa la estimación de la entropía del acelerador, considerando el sketch de conteo de frecuencia, el arreglo de colas, las aproximaciones de logaritmo y división. Al final, se evalúa el rendimiento, la utilización de recursos y el consumo energético del acelerador de hardware.

5.1.1. Base de datos usadas

Se utilizan siete trazas de flujo de red reales. Una traza es proveniente del servidor de Mendeley Data Research Network [73], un repositorio basado en la nube para almacenar, compartir y encontrar datos. Seis trazas son provenientes del *Center for Applied Internet Data Analysis* (CAIDA) [74]; dentro de los cuales cinco trazas pertenecen al enlace troncal Equinix-Chicago y una corresponde al enlace Equinix-San José. Estas trazas contienen encabezados de paquetes anonimizados en formato pcap y tienen una duración de 60 segundos. En la Tabla 5.1, se presenta la información detallada de las doce trazas ocupados en las evaluaciones experimentales. Para cada uno de las trazas se presentan la cantidad total de elementos M, la cantidad de elementos distintos N y la entropía real normalizada para todo el dataset H_{norm}.

Tabla 5.1: Información detallada de las trazas de tráfico de red utilizados en los experimentos.

Trazas	M	N	H_{norm}
Mendeley	2,668,026	101,833	0.271
Chicago-20150219	15,962,528	252,239	0.593
Chicago-20160121	31,197,995	135,269	0.647
Chicago-20110608	27,046,414	393,237	0.694
Sanjose-20081016	20,919,376	334,579	0.701
Chicago-20080515	12,242,152	199,412	0.757
Chicago-20080319	3,938,619	167,768	0.804

En la Tabla 5.1, se verifica que la cantidad de elementos distintos para cada uno de las trazas es muy elevada, lo que implicaría un uso excesivo de memoria para almacenar estos mismos elementos. Encontrar los K elementos más frecuentes, top-K en un flujo de datos o stream de datos, permite estimar la entropía con un uso mínimo de memoria computacionalmente hablando, así pues los resultados serán una estimación, intentando minimizar el posible error.

5.1.2. Resultados del enfoque 1

El primer experimento consiste en evaluar el impacto del valor de K en la estimación de la entropía. Recordando que el primer enfoque consiste en estimar la entropía empírica utilizando los K elementos más frecuentes. Esto implica tener que conocer el valor K de antemano. Desta forma, se realiza el experimento considerando los siguientes valores de $K = \{1024, 2048, 4096, 6144, 8192, 10240\}$. Las Tabla 5.2 y 5.3, muestran los resultados de la simulación, donde K corresponde al número de las direcciones IP distintas más frecuentes, H_{norm} se refiere a la entropía real normalizada calculada según la ecuación (2.9), $\hat{H}_{norm}$ se refiere a la entropía normalizada de los top-K, calculada según la ecuación (3.2), E_{est} se refiere al error absoluto dado por la ecuación (5.1) y E_{r_est} se refiere al error relativo dado por la ecuación (5.2).

$$E_{est} = |H_{norm} - \hat{H}_{norm}| \tag{5.1}$$

$$E_{r\_est} = \frac{|H_{norm} - \hat{H}_{norm}|}{H_{norm}}, \tag{5.2}$$

Tabla 5.2: Estimación de la entropía normalizada y su error absoluto E_{est}, en función de K usando el enfoque dado por la ecuación (3.2).

K	Mendeley H_{norm}=0.271		Chicago-20150219 H_{norm}=0.593		Chicago-20160121 H_{norm}=0.647		Chicago-20110608 H_{norm}=0.694		Sanjose-20081016 H_{norm}=0.701		Chicago-20080515 H_{norm}=0.757		Chicago-20080319 H_{norm}=0.804	
	$\hat{H}_{norm}$	E_{est}	$\hat{H}_{norm}$	E_{est}	$\hat{H}_{norm}$	E_{est}	$\hat{H}_{norm}$	E_{est}	$\hat{H}_{norm}$	E_{est}	$\hat{H}_{norm}$	E_{est}	$\hat{H}_{norm}$	E_{est}
1024	0.193	0.078	0.710	0.117	0.889	0.242	0.911	0.217	0.861	0.160	0.918	0.161	0.947	0.143
4096	0.213	0.058	0.710	0.117	0.840	0.193	0.885	0.191	0.839	0.138	0.907	0.150	0.921	0.117
6144	0.221	0.050	0.703	0.110	0.819	0.172	0.873	0.179	0.830	0.129	0.901	0.144	0.913	0.109
8192	0.226	0.045	0.698	0.105	0.804	0.157	0.863	0.169	0.823	0.122	0.896	0.139	0.908	0.104
10240	0.230	0.041	0.692	0.099	0.791	0.144	0.855	0.161	0.818	0.117	0.891	0.134	0.903	0.099

Tabla 5.3: Estimación de la entropía normalizada y su error relativo E_{r_est}, en función de K usando el enfoque dado por la ecuación (3.2).

K	Mendeley H_{norm}=0.271		Chicago-20150219 H_{norm}=0.593		Chicago-20160121 H_{norm}=0.647		Chicago-20110608 H_{norm}=0.694		Sanjose-20081016 H_{norm}=0.701		Chicago-20080515 H_{norm}=0.757		Chicago-20080319 H_{norm}=0.804	
	$\hat{H}_{norm}$	E_{r_est}	$\hat{H}_{norm}$	E_{r_est}	$\hat{H}_{norm}$	E_{r_est}	$\hat{H}_{norm}$	E_{r_est}	$\hat{H}_{norm}$	E_{r_est}	$\hat{H}_{norm}$	E_{r_est}	$\hat{H}_{norm}$	E_{r_est}
1024	0.193	0.288	0.710	0.197	0.889	0.374	0.911	0.313	0.861	0.228	0.918	0.213	0.947	0.178
2048	0.213	0.214	0.710	0.197	0.840	0.298	0.885	0.275	0.839	0.197	0.907	0.198	0.921	0.146
4096	0.221	0.185	0.703	0.185	0.819	0.266	0.873	0.258	0.830	0.184	0.901	0.190	0.913	0.136
8192	0.226	0.166	0.698	0.177	0.804	0.243	0.863	0.244	0.823	0.174	0.896	0.184	0.908	0.129
10240	0.230	0.151	0.692	0.167	0.791	0.223	0.855	0.232	0.818	0.167	0.891	0.177	0.903	0.123

Las tablas 5.2 y 5.3, muestran los resultados de la estimación de entropía normalizada y sus errores absolutos y relativos, en función de K, calculado según la solución propuesta en el primer enfoque que está definido por la ecuación (3.2). Los resultados muestran que para el método propuesto usando diferentes valores de K que varían entre 1024 y 10240, el error de estimación disminuye cuando se consideran más elementos en la estimación, y el error es mayor cuando el conjunto de datos contiene más elementos distintos. Según el análisis, se define $K = 10240$ para la implementación en hardware, lo que proporciona como resultado un error relativo menor a 0,232 en todos los conjuntos de datos. En la tabla 5.4 se presenta la tabla resumida.

Tabla 5.4: Reporte de la estimación de la entropía, error absoluto y error relativo de los top-10240, según el primer enfoque .

Trazas	H_{norm}	$\hat{H}_{norm}$	E_{est}	E_{r_est}
Mendeley	0.271	0.230	0.041	0.151
Chicago-20150219	0.593	0.692	0.099	0.167
Chicago-20160121	0.647	0.791	0.144	0.223
Chicago-20110608	0.694	0.855	0.161	0.232
Sanjose-20081016	0.701	0.818	0.117	0.167
Chicago-20080515	0.757	0.891	0.134	0.177
Chicago-20080319	0.804	0.903	0.099	0.123

En seguida se evalúa el impacto del uso de un sketch de conteo para estimar el valor de la frecuencia m_i del elemento de entrada e. Para este experimento se han considerado tres sketches distintos: CS, CM y CM-CU, y seis tamaños para cada sketch con $w \in \{8192, 16384, 32768\}$ y $d \in \{4, 9\}$. La Tabla 5.5, presentan los resultados de la simulación de este experimento. Donde E_{est} se refiere al error absoluto dado por la ecuación (5.1), E_{est_s} se refiere al error absoluto de estimación de entropía según el método propuesto en el enfoque 1 basado en la ecuación 3.2, considerando los sketches para la estimación de frecuencia y K=10240. Por $\hat{H}_{norm}$, se refiere a la entropía normalizada de los top-K, calculada según la ecuación (3.2) sin sketches.

Tabla 5.5: Error absoluto de estimación de entropía utilizando la ecuación (3.2) como referencia, para tres tipos de sketches y seis tamaños.

Traza: H_{norm}	w	E_{est_s}					
		CS		CM		CM-CU	
		$d=4$	$d=9$	$d=4$	$d=9$	$d=4$	$d=9$
Mendeley	8K	0.004	0.015	0.024	0.008	0.039	0.020
$\hat{H}_{norm}=0.230$	16K	0.039	0.005	0.041	0.023	**0.012**	0.006
$E_{est}=0.041$	32K	0.018	0.001	0.013	0.005	0.003	0.001
Chicago-20150219	8K	0.177	0.037	0.100	0.056	0.045	0.008
$\hat{H}_{norm}=0.692$	16K	0.112	0.009	0.039	0.019	**0.011**	0.001
$E_{est}=0.099$	32K	0.056	0.001	0.012	0.005	0.002	0.000
Chicago-20160121	8K	0.119	0.024	0.051	0.018	0.029	0.003
$\hat{H}_{norm}=0.791$	16K	0.073	0.004	0.015	0.004	**0.006**	0.000
$E_{est}=0.144$	32K	0.033	0.001	0.003	0.001	0.001	0.000
Chicago-20110608	8K	0.107	0.04	0.078	0.044	0.045	0.008
$\hat{H}_{norm}=0.855$	16K	0.084	0.012	0.035	0.015	**0.014**	0.001
$E_{est}=0.161$	32K	0.052	0.002	0.011	0.005	0.003	0.001
Sanjose-20081016	8K	0.121	0.038	0.094	0.061	0.044	0.009
$\hat{H}_{norm}=0.818$	16K	0.088	0.011	0.044	0.023	**0.013**	0.001
$E_{est}=0.117$	32K	0.051	0.002	0.015	0.007	0.002	0.000
Chicago-20080515	8K	0.072	0.026	0.058	0.035	0.028	0.005
$\hat{H}_{norm}=0.891$	16K	0.053	0.007	0.026	0.010	**0.008**	0.000
$E_{est}=0.134$	32K	0.030	0.001	0.007	0.002	0.001	0.000
Chicago-20080319	8K	0.062	0.024	0.059	0.042	0.025	0.006
$\hat{H}_{norm}=0.903$	16K	0.044	0.008	0.030	0.015	**0.007**	0.001
$E_{est}=0.099$	32K	0.025	0.001	0.010	0.004	0.001	0.000

En la Tabla 5.5, se presentan los errores absolutos de estimación de la entropía considerando los tres sketches para seis tamaños distintos, según el enfoque 1 definido por la ecuación 3.2. Los resultados muestran que el valor del error absoluto se reduce a medida que se aumenta la dimensión del sketch. Adicionalmente también se verifica que la implementación del CM-CU 9x8K, 9x16K, 9x32K, proporciona estimaciones más precisas que las proporcionadas por CS y CM, para las mismas dimensiones del sketch. Por lo tanto, se elige el CM-CU sketch para usar

en la estimación de la frecuencia m_i de los elementos e en el método propuesto. Aunque el sketch CM-CU 9x32K presenta el error más bajo, se verifica que con un sketch de menor tamaño, se puede obtener resultados de entropía aproximados. Por ejemplo, se verifica que con el uso de un sketch CM-CU 4x16K incorpora un error de 0.013 en comparación con el sketch más grande con una reducción de ocupación de memoria en 25 %. Para la implementación en hardware se usan los siguientes parámetros del sketch $d = 4$ y $w = 16$K. Para esos valores del sketch se resalta el error absoluto en la tabla.

En seguida se incorpora el arreglo de colas y se realiza la simulación para verificar el impacto del uso del arreglo de colas en la implementación del CM-CU 4x16K.

Tabla 5.6: Reporte del error absoluto de estimación de la entropía considerando el sketch y arreglo de colas.

Trazas	E_{est}	E_{est_s}	$E_{est\_s\_pq}$
Mendeley	0.041	0.012	0.000
Chicago-20150219	0.099	0.011	0.000
Chicago-20160121	0.144	0.006	0.000
Chicago-20110608	0.161	0.014	0.002
Sanjose-20081016	0.117	0.013	0.000
Chicago-20080515	0.134	0.008	0.000
Chicago-20080319	0.099	0.007	0.001

Los resultados en la tabla 5.6, muestran que al incorporar el arreglo de colas en la implementación del CM-CU 4x16K, este agrega un error máximo de 0.002 en la estimación de la entropía.

Posteriormente se realiza la simulación del acelerador considerando todos los parámetros definidos anteriormente.

Los resultados en la tabla 5.7 muestra, el error en la implementación del hardware con respecto a la implementación del software. Este error incluye operaciones de punto fijo en log_2 y cálculo de Newton-Raphson. Los resultados muestran que el acelerador agrega un error máximo de 0.006 en la estimación de la entropía. Esto demuestra que la arquitectura implementada calcula una entropía normalizada con un error mínimo en comparación con la implementación de software.

Tabla 5.7: Reporte de la estimación de la entropía usando el acelerador.

Trazas	E_{est}	E_{est_s}	$E_{est\_s\_pq}$	E_{acc}
Mendeley	0.041	0.012	0.000	0.000
Chicago-20150219	0.099	0.011	0.000	0.001
Chicago-20160121	0.144	0.006	0.000	0.006
Chicago-20110608	0.161	0.014	0.002	0.002
Sanjose-20081016	0.117	0.013	0.000	0.003
Chicago-20080515	0.134	0.008	0.000	0.002
Chicago-20080319	0.099	0.007	0.001	0.002

En la tabla 5.8, se presentan los resultados de la utilización de recursos.

Tabla 5.8: Utilización de recursos FPGA por módulo

Módulo	LUTs	Registros	BRAMs	DSPs
Countmin-CU	1,175	1,105	64	30
arreglo PQ	781	811	10	0
Entropía	637	765	0.5	17
Misc	14	21	0	0
Total	2,607	2,730	74.5	47
Disponible	230,400	460,800	312	1,728
Porcentaje (%)	1.13	0.59	23.88	2.72

La implementación de hardware se ejecuta en una FPGA Xilinx Zynq UltraScale+ MPSoC ZCU102. La tabla 5.8 resume la utilización de recursos por módulo, donde los BRAM presentan la mayor utilización con un 23.88 %. El módulo Countmin-CU Sketch es responsable de la mayor parte de los DSP con 30 DSP para el cálculo de MurmurHash3. El módulo de entropía usa las DSP restantes en el módulo de Newton-Raphson de iteraciones (8), $\log_2$ LUT (2) y cálculo de entropía (5). El módulo de sketch Countmin-CU también utiliza la mayoría de los BRAM, utilizando 16 BRAM por bucket con un total de 64 BRAM, mientras que el arreglo de colas de prioridad utilizan 10 BRAM para almacenar los K elementos más frecuentes. El uso de todos los recursos es menos de 24 % lo que nos permite usar varias instancias de esta arquitectura o usarla en combinación con otro proceso.

5.2. Enfoque 2

En esta sección se presentan los resultados obtenidos en los experimentos realizados para evaluar el método propuesto en el segundo enfoque, que es dado por la ecuación (3.2). Primeramente se describen las bases de datos ocupadas para realizar todos los experimentos, y luego se presentan las informaciones detalladas de esas trazas de red, tales como; nombre de identificación, cantidad total de paquetes, cantidad total de elementos distintos y el valor de la entropía exacta de todos elementos de la traza, calculada según la ecuación (2.9). Luego se presentan los resultados de la evaluación de la entropía en función del valor de K. Luego con el valor de K definido, se presentan los resultados de la evaluación de la estimación de la entropía considerando el sketch de conteo de elementos distintos. A continuación se evalúa el impacto de incorporar los distintos sketches de conteo de frecuencia para diferentes tamaños. Posteriormente con los parámetros de los sketches ya definidos para usar en la implementación del hardware, se evalúa la incorporación del arreglo de colas. Después se evalúa la estimación de la entropía del acelerador, considerando el sketch de cardinalidad y conteo de frecuencia, el arreglo de colas, las aproximaciones de logaritmo y división. Al final, se evalúa el rendimiento, la utilización de recursos y el consumo energético del acelerador de hardware.

5.2.1. Base de datos usadas

Se utilizan doce trazas de flujo de red real para realizar la evaluación experimental del método propuesto en este enfoque. De estas trazas; una traza es proveniente del servidor de Mendeley Data Research Network [73], un repositorio basado en la nube para almacenar, compartir y encontrar datos. Cinco trazas de 15 minutos pertenecen al MAWILab [75], una base de datos que ayuda a los investigadores a evaluar los métodos de detección de anomalías en el tráfico de red. Seis trazas son provenientes del *Center for Applied Internet Data Analysis* (CAIDA) [74]; dentro de los cuales cinco trazas pertenecen al enlace troncal Equinix-Chicago y una corresponde al enlace Equinix-San José. Estas trazas contienen encabezados de paquetes anonimizados en formato pcap y tienen una duración de 60 segundos. En la Tabla 5.9, se presentan la información detallada de lss doce trazas ocupados en las evaluaciones experimentales. Para cada una de los trazas se presentan la cantidad total de elementos M, la cantidad de elementos distintos N y los valores de la entropía real normalizada para todo el dataset H_{norm}.

Tabla 5.9: Información detallada de las trazas de tráfico de red utilizados en los experimentos.

Trazas	M	N	H_{norm}
Mendeley	2,668,026	101,833	0.271
Chicago-20150219	15,962,528	252,239	0.593
Chicago-20160121	31,197,995	135,269	0.647
Chicago-20110608	27,046,414	393,237	0.694
Sanjose-20081016	20,919,376	334,579	0.701
Chicago-20080515	12,242,152	199,412	0.757
Chicago-20080319	3,938,619	167,768	0.804
Mawi-20201400	119,923,870	5,394,529	0.366
Mawi-20191400	62,478,145	4,633,485	0.406
Mawi-20181400	76,066,888	4,674,492	0.404
Mawi-20171400	115,486,661	4,145,741	0.437
Mawi-20161400	94,738,984	5,083,756	0.448

5.2.2. Resultados del enfoque 2

El primer experimento consiste en evaluar el impacto del valor de K en la estimación de la entropía, según el método propuesto en el enfoque 2. Se realiza el experimento usando diferentes valores de $K = \{1024, 2048, 4094, 8192, 16384\}$. La Figura 5.1 muestra el resultado de la simulación, considerando los recuentos de frecuencia exactos del los elementos top-K y suponiendo que la frecuencia de los elementos menos frecuentes $N - K$ se distribuyen uniformemente. Donde K corresponde al número de las top-K direcciones de IP y E_{r_est} se refiere al error relativo dado por la ecuación 5.4.

$$E_{est} = |H_{norm} - \hat{H}_{norm}| \tag{5.3}$$

$$E_{r\_est} = \frac{|H_{norm} - \hat{H}_{norm}|}{H_{norm}} \times 100, \tag{5.4}$$

Donde H_{norm} es la entropía exacta normalizada de la traza dada por la ecuación (2.9), y $\hat{H}_{norm}$ es la estimación de la entropía normalizada dada por la ecuación (3.6) utilizando los valores exactos de la cardinalidad N y los recuentos de frecuencia m_i.

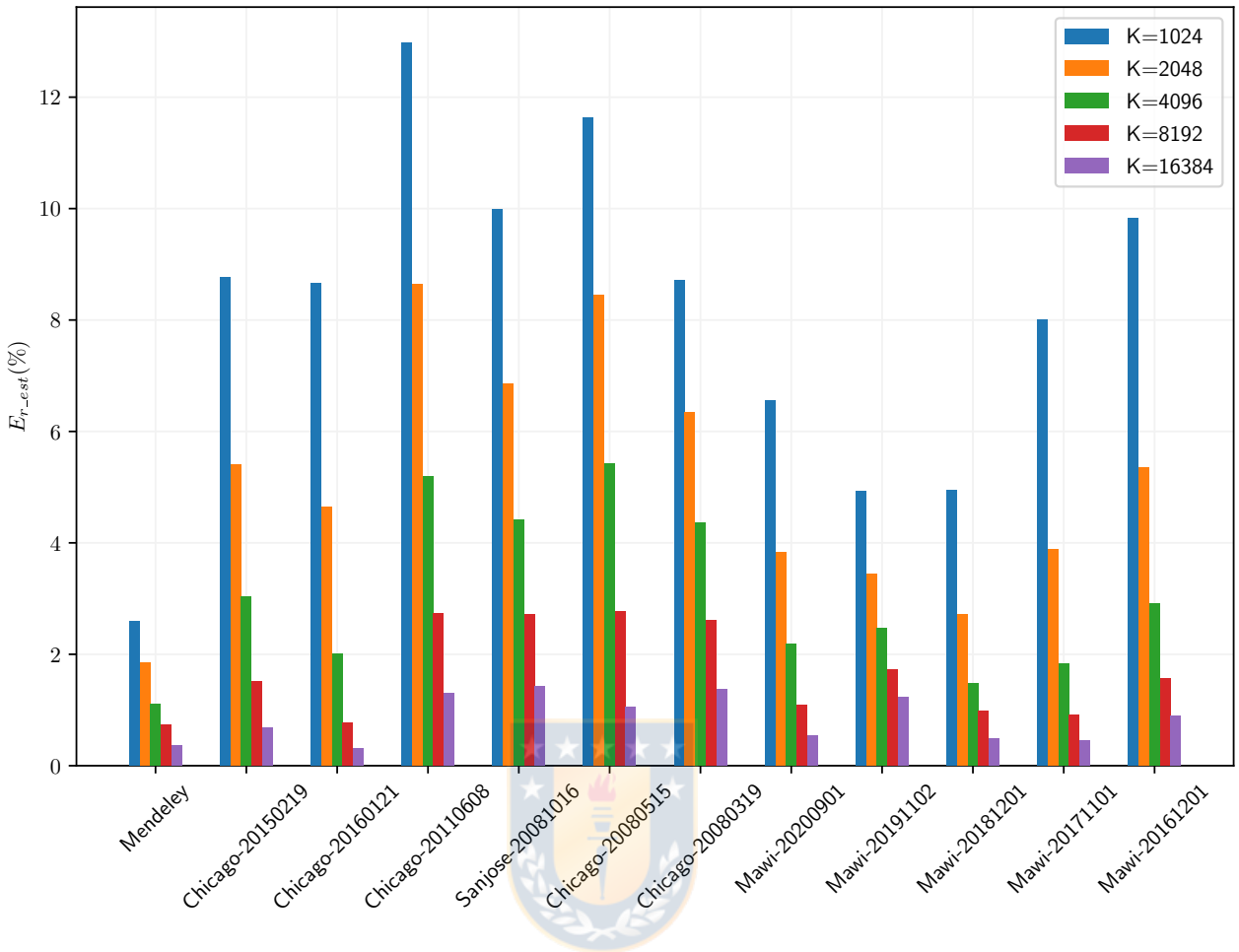

Figura 5.1: Error relativo de estimación de entropía según la ecuación (3.6) con respecto a la ecuación (2.9) en función de K, considerando los valores exactos de N y m_i.

Los resultados en la Figura 5.1, muestran que el error de estimación de la entropía disminuye con el aumento del valor de K. Además, se verifica que para $K \geq 8192$, el $E_{r\_est} < 3\,\%$ para todas las trazas y su valor medio es $1.67\,\%$. Teniendo en cuenta que un error de entropía normalizado inferior al $3\,\%$ es suficiente para discriminar entre la entropía del tráfico normal y los diferentes tipos de ataques DoS y DDoS [76], se elige $K = 8192$, con $R = 1024$ y $S = 8$, para la implementación.

En el segundo experimento se evalúa el impacto del HLL sketch para estimar la cardinalidad de la traza. En este experimento, se calcula la entropía estimada usando la ecuación (3.6) con $K = 8192$ y se estima N con un HLL sketch usando para el parámetro de precisión los siguientes valores $p = \{10, 11, 12, 13, 14\}$. La Figura 5.2, muestra el resultado de la simulación, donde $E_{r\_est\_HLL}$, es el error relativo de la estimación de la entropía con respecto a $\hat{H}_{norm}$, que

viene dado por la ecuación (5.5):

$$E_{r\_est\_HLL} = \frac{|\hat{H}_{norm} - \hat{H}_{norm\_HLL}|}{\hat{H}_{norm}}, \tag{5.5}$$

Donde $\hat{H}_{norm\_HLL}$ es la entropía normalizada calculada por la ecuación (3.6) utilizando el valor de N estimado por el HLL sketch.

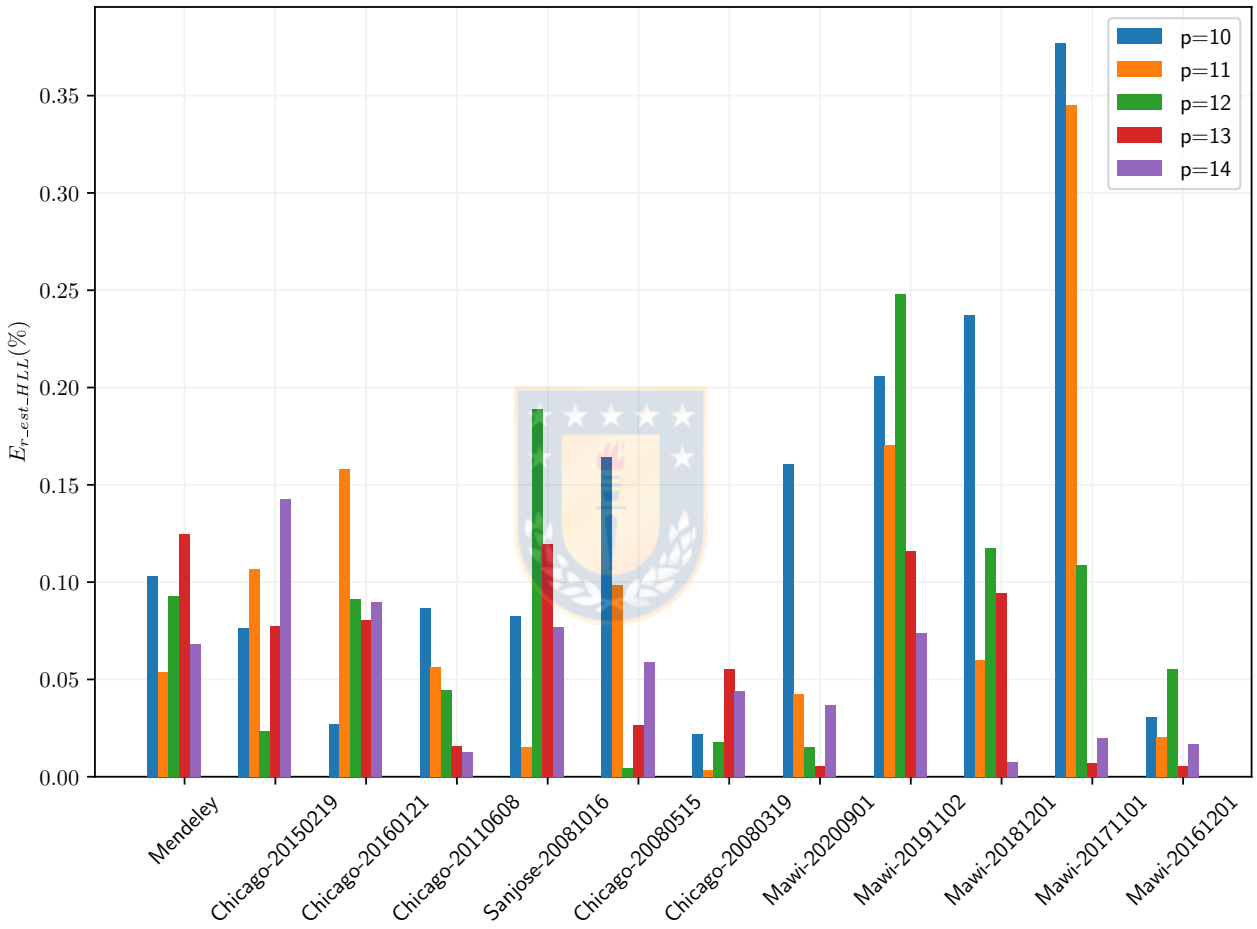

Figura 5.2: Error relativo con respecto a la estimación de entropía según la ecuación (3.6) cuando se usa un HLL sketch para estimar N, en función del parámetro de precisión p.

La Figura 5.2 muestra los resultados de la simulación al considerar la incorporación del HLL sketch en la implementación del método. Se presentan los resultados de la estimación de la entropía, según el método propuesto, considerando distintos valores de precisión del HLL sketch. Los resultados muestran que en general el error de la estimación de la entropía mejora a medida que se aumenta el valores de p. En todos los casos el error es menor que $0.4\,\%$. Se verifica que al considerar para la implementación en hardware un valor de $p = 13$, el error introducido por el HLL sketch es menor a $0.15\,\%$. Para este valor de p, se implementará un sketch de 8192×5-bit se puede implementar con menos de dos bloques de memoria BRAM en una FPGA.

En seguida, se evalúa el uso de un sketch de conteo para estimar el valor de la frecuencia m_i en la ecuación (3.5). Se han considerado tres sketches diferentes: CS, CM y CM-CU, y cuatro tamaños para cada sketch, con $w \in \{8192, 16384\}$ y $d \in \{8, 16\}$. Se evaluó el error relativo que los sketches de conteo y cardinalidad agregan a la estimación de entropía $\hat{H}_{norm}$ usando $K = 8192$, como lo define la ecuación (5.6):

$$E_{r\_est\_s} = \frac{|\hat{H}_{norm} - \hat{H}_{norm\_s}|}{\hat{H}_{norm}}, \tag{5.6}$$

Donde $\hat{H}_{norm\_s}$ es la entropía normalizada estimada por la ecuación (3.6) usando frecuencias m_i estimado por el sketch de conteo y la cardinalidad N estimado por el HLL sketch con $p = 13$.

La tabla 5.10 muestra los valores de $E_{r\_est\_s}$ para todas las trazas del experimento realizado. La primera columna enumera la traza y su entropía estimada $\hat{H}_{norm}$ usando recuentos exactos y $K = 8192$. El resto de las columnas muestran E_{est_s} para cada tipo y tamaño del sketch.

Tabla 5.10: Error relativo de estimación de entropía utilizando la ecuación (3.6) como referencia, para tres tipos de sketches y cuatro tamaños.

Trazas: H_{norm}	w	$E_{r\_est\_s}(\%)$					
		CS		CM		CM-CU	
		$d = 8$	$d = 16$	$d = 8$	$d = 16$	$d = 8$	$d = 16$
Mendeley	8K	1.97	0.81	6.46	4.54	1.28	0.88
$\hat{H}_{norm} = 0.273$	16K	0.84	0.38	1.66	1.10	0.45	0.33
Chicago-20150219	8K	8.97	1.81	7.13	4.78	0.55	0.26
$\hat{H}_{norm} = 0.602$	16K	2.04	0.41	1.90	1.39	0.15	0.12
Chicago-20160121	8K	1.39	1.44	2.75	1.49	0.31	0.13
$\hat{H}_{norm} = 0.652$	16K	1.68	0.28	0.58	0.36	0.10	0.09
Chicago-20110608	8K	10.10	3.14	9.72	5.81	0.64	0.15
$\hat{H}_{norm} = 0.713$	16K	3.85	0.59	2.28	1.58	0.08	0.04
Sanjose-20081016	8K	10.33	2.45	11.01	7.29	0.72	0.34
$\hat{H}_{norm} = 0.720$	16K	2.72	0.65	2.85	2.06	0.21	0.17
Chicago-20080515	8K	9.82	2.48	8.61	4.82	0.49	0.14
$\hat{H}_{norm} = 0.778$	16K	2.60	0.58	1.72	1.06	0.07	0.04
Chicago-20080319	8K	6.71	2.02	10.15	6.47	0.49	0.22
$\hat{H}_{norm} = 0.825$	16K	2.05	0.61	2.34	1.47	0.12	0.08
Mawi-20200901	8K	1.58	2.89	9.40	7.37	1.16	0.46
$\hat{H}_{norm} = 0.370$	16K	6.20	0.39	3.24	2.90	0.18	0.12
Mawi-20191102	8K	12.85	1.47	12.72	10.74	1.58	1.03
$\hat{H}_{norm} = 0.413$	16K	3.10	0.19	4.52	4.02	0.30	0.17
Mawi-20181201	8K	4.00	1.74	10.02	8.37	1.18	0.71
$\hat{H}_{norm} = 0.408$	16K	4.11	0.15	3.57	3.28	0.21	0.12
Mawi-20171101	8K	4.12	2.83	6.75	4.95	1.10	0.41
$\hat{H}_{norm} = 0.441$	16K	6.34	0.29	2.18	1.96	0.20	0.14
Mawi-20161201	8K	3.41	4.31	12.38	9.66	1.95	0.83
$\hat{H}_{norm} = 0.455$	16K	9.59	0.52	4.26	3.84	0.39	0.27

Los resultados en la tabla 5.10 muestran que, para los valores de los parámetros definidos para realizar la simulación de este experimento, la precisión de los sketches depende en gran medida

del tamaño del sketch. En el caso de CM y CM-CU, la dependencia es más fuerte con respecto a w, que determina el número de columnas en la matriz, que d, lo cuál establece el número de filas. La sobreestimación del recuento de CM da como resultado estimaciones de entropía que en la mayoría de los casos son menos precisas que CS y consistentemente peores que CM-CU, para el mismo tamaño de sketch. La precisión de estimación lograda con CM-CU es consistentemente mejor que CS y CM usando el mismo tamaño. Además, el error de estimación de entropía de CM-CU obtenido usando $d = 8$ está cerca del error con $d = 16$. Con base en estos resultados, se decidió usar un CM-CU sketch con $w = 16384$ y $d = 8$, lo que agrega un promedio de 0.2 % a la estimación de la ecuación (3.6), y logra el mejor compromiso entre precisión y requisitos de memoria.

En seguida, se evalúa el impacto de las técnicas de implementación utilizadas en el acelerador. Se realizó el experimento de la implementación del acelerador considerando los siguientes parámetros: un HLL sketch de 8K celdas con $p = 13$ y $b = 5$, un CM-CU sketch con $w = 16384$ y $d = 8$, y un arreglo de PQ's con $R = 1024$ y $S = 8$. Los tres módulos utilizan funciones hash Murmur3 de 32 bits. Los contadores de flujo en el CM-CU sketch y la PQ tienen un ancho de 25 bits, como lo requieren las trazas grandes de MAWILab. La tabla 5.11 resume los resultados de estimación de entropía normalizada. Donde $\hat{H}_{norm}$ es la estimación de entropía calculada según la ecuación (3.6). $\hat{H}_{norm\_acc}$ es la estimación de entropía calculada por el acelerador e incluye las implementaciones de hardware del arreglo de PQ's y el CM-CU sketch para calcular la frecuencia de los 8192 elementos más frecuentes y el HLL sketch para estimar N. Además, $\hat{H}_{norm\_acc}$ también incluye el efecto de los divisores de Newton-Raphson y el cálculo de la función logarítmica usando una LUT e interpolación lineal.

Tabla 5.11: Resumen de la estimación de entropía obtenida por la ecuación (3.6) y el acelerador.

Trazas	H_{norm}	$\hat{H}_{norm}$	$\hat{H}_{norm\_acc}$
Mendeley	0.271	0.273	0.272
Chicago-20150219	0.593	0.602	0.603
Chicago-20160121	0.647	0.652	0.650
Chicago-20110608	0.694	0.713	0.713
Sanjose-20081016	0.701	0.720	0.720
Chicago-20080515	0.757	0.778	0.780
Chicago-20080319	0.804	0.825	0.826
Mawi-20200901	0.366	0.370	0.364
Mawi-20191102	0.406	0.413	0.410
Mawi-20181201	0.404	0.408	0.405
Mawi-20171101	0.437	0.441	0.442
Mawi-20161201	0.448	0.455	0.453

En la tabla 5.11 se presentan los resultados de estimación de entropía normalizada. Se verifica que al considerar todos los parámetros mencionados anteriormente en la implementación del acelerador y usarlo para la estimación de entropía, se introduce un error de 0.003 en los resultados del acelerador en comparación con los resultados de la simulación en software del método.

En seguida, se presenta en la Tabla 5.12 para las 12 trazas, los errores absolutos y relativos para la estimación de entropía usando el método propuesto, calculada según las ecuaciones (5.7) y (5.8). También presenta los errores absolutos y relativos para la entropía estimada por el acelerador, calculada como:

$$E_{est\_acc} = |\hat{H}_{norm\_acc} - H_{norm}| \tag{5.7}$$

$$E_{r\_est\_acc}(\%) = \frac{|\hat{H}_{norm\_acc} - H_{norm}|}{H_{norm}} \times 100, \tag{5.8}$$

donde H_{norm} es el valor exacto de la entropía empírica normalizada calculada por la ecuación (2.9).

Tabla 5.12: Resumen de los errores de estimación absolutos y relativos obtenidos por la ecuación (3.6) y el acelerador.

Trazas	H_{norm}	E_{est}	$E_{r\_est}(\%)$	E_{acc}	$E_{r\_acc}(\%)$
Mendeley	0.271	0.002	0.74	0.001	0.37
Chicago-20150219	0.593	0.008	1.35	0.010	1.69
Chicago-20160121	0.647	0.005	0.77	0.003	0.46
Chicago-20110608	0.694	0.019	2.74	0.019	2.74
Sanjose-20081016	0.701	0.019	2.71	0.019	2.71
Chicago-20080515	0.757	0.021	2.77	0.023	3.04
Chicago-20080319	0.804	0.021	2.61	0.022	2.74
Mawi-20200901	0.366	0.005	1.37	0.002	0.55
Mawi-20191102	0.406	0.006	1.48	0.004	0.99
Mawi-20181201	0.404	0.004	0.99	0.001	0.25
Mawi-20171101	0.437	0.004	0.92	0.003	0.69
Mawi-20161201	0.448	0.006	1.34	0.005	1.12

Los resultados muestran que el error relativo de estimación del método varia entre 0.74 % y 2.77 %, con un valor medio de 1.65 % y una desviación estándar de 0.82 %. El error relativo en la traza de Mendely es de 0.74 %, el error medio en las trazas de CAIDA es de 2.16 % y en las trazas de MAWILab es de 1.22 %. En el acelerador, el error de estimación varia entre 0.25 % y 3.04 %, con una media de 1.45 % y una desviación estándar de 1.08 %. En muchos casos, el error del acelerador es en realidad menor que el del método porque los errores de frecuencia introducidos por el CM-CU sketch reducen el valor de entropía, que el método sobreestima constantemente. El error de entropía relativa del acelerador en la traza de Mendely es de 0.37 %, el error medio en las trazas de CAIDA es de 2.23 % y en las trazas de MAWILab es de 0.72 %. Al observar el valor de la entropía absoluta, tanto el método como el acelerador muestran un error de estimación inferior a 0.023 para todas las trazas. El error de estimación absoluto medio del método es 0.010 y para el acelerador es 0.009, con desviaciones estándar de 0.008 y 0.009, respectivamente.

En la tabla 5.13, se presentan los resultados de la utilización de recursos.

Tabla 5.13: Utilización de recursos FPGA por módulo

Módulo	LUTs	Registros	BRAMs	DSPs
Countmin-CU	2,854	2,991	128	96
arreglo PQ	526	812	12	0
HLL	643	671	1.5	18
Entropía	643	844	0.5	14
Misc	31	87	0	0
Total	4,697	5,405	142	128
Disponible	230,400	460,800	312	1,728
Porcentaje (%)	2.04	1.17	45.51	7.41

La implementación de hardware se ejecuta en una FPGA Xilinx Zynq UltraScale+ ZCU102. La tabla 5.13 resume la utilización de recursos por módulo. El CM-CU sketch es el módulo más grande: utiliza 128 BRAM (aproximadamente 512 KB) para implementar la matriz del sketch. Aunque el sketch usa contadores de 25 bits, los bloques BRAM se pueden configurar solo para 18 o 36 bits, por lo que cada fila del sketch usa 16 $1K \times 36$-bit BRAM. La mayoría de los registros en CM-CU se utilizan como registros de canalización en sus 8 instancias de MurmurHash3, y los segmentos de DSP se utilizan en las operaciones de la función hash, para incrementar los contadores de sketch y en los comparadores que seleccionan el valor mínimo. El arreglo de PQ's usa 12 BRAM (aproximadamente 48 KB) de almacenamiento, donde cada una de las memorias S usa 1.5 BRAM para almacenar 1024 tuplas que contienen el contador de 25 bits y una etiqueta de 22 bits. La mayoría de los registros individuales se utilizan para almacenar datos temporales en la canalización. La cola comparte la función hash con la primera fila del sketch CM-CU, por lo que no utiliza segmentos de DSP. El módulo HLL usa 1.5 BRAM para su celda y 18 DSP para los divisores de Newton-Raphson y la aproximación de función logarítmica. El módulo de estimación de entropía utiliza los 14 DSP restantes en sus divisores y cálculo de logaritmos. En general, todo el diseño utiliza una fracción de los recursos disponibles en el chip, donde los BRAM muestran el uso más alto del 45.5 %. Por lo tanto, se encuentran disponibles importantes recursos de hardware para expandir la arquitectura, implementar múltiples instancias del acelerador o agregar cálculos adicionales para el análisis del tráfico de la red.

Tabla 5.14: Consumo de energía FPGA por módulo

Módulo	Power (W)
Countmin-CU	0.90
Arreglo PQ	0.13
HLL	0.11
Entropía	0.09
Misc	0.01
Total	1.24

La tabla 5.14 muestra el consumo de energía del núcleo del acelerador según *Xilinx Power Estimator*. Al funcionar a 400 MHz, el núcleo consume 1.24 mW, la mayor parte de los cuales son consumidos por los bloques BRAM del CM-CU sketch. Los segmentos de DSP también aumentan el consumo de energía, especialmente en los módulos CM-CU y HLL.

Capítulo 6. Conclusiones y trabajos futuros

6.1. Conclusiones

En el presente trabajo, se han presentado dos métodos y implementados en arquitecturas de hardware para estimar la entropía empírica sobre flujos de datos utilizando algoritmos de streaming de datos. El primer método utiliza una estructura de datos basada en sketch, un Countmin con actualizaciones conservadoras para mantener una estimación de la frecuencia de todos los elementos en el flujo, y una cola de prioridad para almacenar los elementos más frecuentes y sus frecuencias. Se ha implementado la arquitectura del acelerador a nivel de transferencia de registro utilizando el lenguaje de descripción de hardware SystemVerilog. Se han sintetizado el diseño usando Vivado 2018.2 y se implementó en una FPGA Xilinx Zynq UltraScale+ ZCU102. El diseño se ejecuta a 354 MHz y la ruta crítica se debe a un retardo de enrutamiento en la operación de actualización del CM-CU sketch. A esta frecuencia de reloj, el acelerador puede funcionar a una velocidad de línea de al menos 181 Gbps, asumiendo el peor de los casos de paquetes de tamaño mínimo de 64 bytes, mientras disipa una potencia de 551 mW. El segundo y principal método utiliza un Countmin con actualizaciones conservadoras para mantener una estimación de la frecuencia de todos los elementos en el flujo, un HLL sketch para aproximar la cardinalidad, y una cola de prioridad para almacenar los elementos más frecuentes y sus frecuencias. El diseño se ha implementado con éxito en un Xilinx Zynq UltraScale+ MPSoC ZCU102 FPGA. La arquitectura puede estimar la entropía con una tasa mínima de 204 Gbps, mientras que utiliza menos de 23.56 % de recursos de hardware. Esto permite estimar la entropía para múltiples campos en paquetes Ethernet, o incorporar el núcleo a otro sistema.

Los resultados reportados, muestran que el método basado en el segundo enfoque presenta mejores resultados que el método basado en el primer enfoque. También se verifica que para el método basado en el segundo y principal enfoque, el error de estimación se introduce principalmente por la caracterización de los elementos menos frecuentes como una distribución uniforme. Aunque el método sobreestima la contribución de estos elementos a la entropía, nos permite calcular la entropía como una expresión simple que depende de la información recopilada para los flujos top-K en el flujo de la red. Además, para K suficientemente grandes, se puede estimar la entropía con un error del 3 % o menos, y un promedio de 1.45 %, para las trazas utilizadas en los experimentos. Los sketches HLL y CM-CU introducen un error medio en la estimación de 0.21 %, el arreglo de PQ's 0.06 % y las aproximaciones de funciones aritméticas suman 0.46 %. El

uso de estas técnicas permite implementar el algoritmo de estimación de entropía completo en el plano de datos utilizando exclusivamente recursos en chip, logrando así un alto rendimiento, baja potencia y baja utilización de recursos.

Para los trabajos futuros se propone estudiar y/o utilizar una otra distribución que permita disminuir el error de estimación que se introduce por la contribución de los elementos menos frecuentes.

Referencias

[1] Javier E Soto, Paulo Ubisse, Cecilia Hernández, and Miguel Figueroa. A hardware accelerator for entropy estimation using the top-k most frequent elements. In *2020 23rd Euromicro Conference on Digital System Design (DSD)*, pages 141–148. IEEE, 2020.

[2] Javier E Soto, Paulo Ubisse, Yaime Fernández, Cecilia Hernández, and Miguel Figueroa. A high-throughput hardware accelerator for network entropy estimation using sketches. *IEEE Access*, 2021.

[3] Ashwin Lall, Vyas Sekar, Mitsunori Ogihara, Jun Xu, and Hui Zhang. Data streaming algorithms for estimating entropy of network traffic. In *ACM SIGMETRICS Performance Evaluation Review*, volume 34, pages 145–156. ACM, 2006.

[4] Arno Wagner and Bernhard Plattner. Entropy based worm and anomaly detection in fast ip networks. In *14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise (WETICE'05)*, pages 172–177. IEEE, 2005.

[5] Graham Cormode and Minos Garofalakis. Sketching probabilistic data streams. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 281–292. ACM, 2007.

[6] Shanmugavelayutham Muthukrishnan et al. Data streams: Algorithms and applications. *Foundations and Trends® in Theoretical Computer Science*, 1(2):117–236, 2005.

[7] Balachander Krishnamurthy, Subhabrata Sen, Yin Zhang, and Yan Chen. Sketch-based change detection: methods, evaluation, and applications. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, pages 234–247. ACM, 2003.

[8] Da Tong and Viktor K Prasanna. Sketch acceleration on fpga and its applications in network anomaly detection. *IEEE Transactions on Parallel and Distributed Systems*, 29(4):929–942, 2017.

[9] Yu-Kuen Lai, Ku-Yeh Shih, Po-Yu Huang, Ho-Ping Lee, Yu-Jau Lin, Te-Lung Liu, and Jim Hao Chen. Sketch-based entropy estimation for network traffic analysis using programmable data plane asics. In *2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pages 1–2. IEEE, 2019.

[10] Rochak Swami, Mayank Dave, and Virender Ranga. Defending DDoS against Software Defined Networks using Entropy. In *2019 4th International Conference on Internet of Things: Smart Innovation and Usages (IoT-SIU)*, pages 1–5. IEEE, 2019.

[11] Lu Zhou, Keshav Sood, and Yong Xiang. ERM: An accurate approach to detect DDoS attacks using entropy rate measurement. *IEEE Communications Letters*, 23(10):1700–1703, 2019.

[12] Xi Wang and Yemeng Jiang. An abnormal traffic discovery way based on exbm mechanism in software defined network. In *2019 4th International Conference on Cloud Computing and Internet of Things (CCIOT)*, pages 6–9. IEEE, 2019.

[13] Shun Otsubo, Sosuke Ito, Andreas Dechant, and Takahiro Sagawa. Estimating entropy production by machine learning of short-time fluctuating currents. *arXiv preprint arXiv:2001.07460*, 2020.

[14] Yan Zhang, Cheng Wen, Changxin Wang, Stoichko Antonov, Dezhen Xue, Yang Bai, and Yanjing Su. Phase prediction in high entropy alloys with a rational selection of materials descriptors and machine learning models. *Acta Materialia*, 185:528–539, 2020.

[15] Qingfeng Wu, Zhijun Wang, Xiaobing Hu, Tao Zheng, Zhongsheng Yang, Feng He, Junjie Li, and Jincheng Wang. Uncovering the eutectics design by machine learning in the al–co–cr–fe–ni high entropy system. *Acta Materialia*, 182:278–286, 2020.

[16] Narges Javidan, Ataollah Kavian, Hamid Reza Pourghasemi, Christian Conoscenti, and Zeinab Jafarian. Data mining technique (maximum entropy model) for mapping gully erosion susceptibility in the gorganrood watershed, iran. In *Gully Erosion Studies from India and Surrounding Regions*, pages 427–448. Springer, 2020.

[17] Anhui Tan, Suwei Shi, Wei-Zhi Wu, Jinjin Li, and Witold Pedrycz. Granularity and entropy of intuitionistic fuzzy information and their applications. *IEEE Transactions on Cybernetics*, 2020.

[18] Devaraju Sellappan and Ramakrishnan Srinivasan. Association rule-mining-based intrusion detection system with entropy-based feature selection: Intrusion detection system. In *Handbook of Research on Intelligent Data Processing and Information Security Systems*, pages 1–24. IGI Global, 2020.

[19] Salvatore Carta, Anselmo Ferreira, Diego Reforgiato Recupero, Marco Saia, and Roberto Saia. A combined entropy-based approach for a proactive credit scoring. *Engineering Applications of Artificial Intelligence*, 87:103292, 2020.

[20] Peng Yue, Yaodong Fan, Jonathan A Batten, and Wei-Xing Zhou. Information transfer between stock market sectors: A comparison between the usa and china. *Entropy*, 22(2):194, 2020.

[21] Chaojie Wang. A sample entropy inspired affinity propagation method for bearing fault signal classification. *Digital Signal Processing*, page 102740, 2020.

[22] Lei Wang, Libo Zhang, Xiaodong Yang, Peiwei Yi, and Hao Chen. Level set based segmentation using local fitted images and inhomogeneity entropy. *Signal Processing*, 167:107297, 2020.

[23] Jesús Galeano-Brajones, Javier Carmona-Murillo, Juan F Valenzuela-Valdés, and Francisco Luna-Valero. Detection and Mitigation of DoS and DDoS Attacks in IoT-Based Stateful SDN: An Experimental Approach. *Sensors*, 20(3):816, 2020.

[24] Eduardo Viegas, Altair Santin, Alysson Bessani, and Nuno Neves. Bigflow: Real-time and reliable anomaly-based intrusion detection for high-speed networks. *Future Generation Computer Systems*, 93:473–485, 2019.

[25] Alka Gupta and Lalit Sen Sharma. Detecting attacks in high-speed networks: Issues and solutions. *Information Security Journal: A Global Perspective*, 29(2):51–61, 2020.

[26] Amritpal Singh, Sahil Garg, Ravneet Kaur, Shalini Batra, Neeraj Kumar, and Albert Y Zomaya. Probabilistic data structures for big data analytics: A comprehensive review. *Knowledge-Based Systems*, 188:104987, 2020.

[27] Thalita Vergilio, Muthu Ramachandran, and Duncan Mullier. Requirements engineering for large-scale big data applications. In *Software Engineering in the Era of Cloud Computing*, pages 51–84. Springer, 2020.

[28] Leszek Rutkowski, Maciej Jaworski, and Piotr Duda. Basic concepts of data stream mining. In *Stream Data Mining: Algorithms and Their Probabilistic Properties*, pages 13–33. Springer, 2020.

[29] Ilham Amezzane, Youssef Fakhri, Mohamed El Aroussi, and Mohamed Bakhouya. Hardware acceleration of svm training for real-time embedded systems: Overview. In *Recent Advances in Mathematics and Technology*, pages 131–139. Springer, 2020.

[30] Shuhao Zhang, Feng Zhang, Yingjun Wu, Bingsheng He, and Paul Johns. Hardware-conscious stream processing: A survey. *ACM SIGMOD Record*, 48(4):18–29, 2020.

[31] Da Tong and Viktor K Prasanna. Sketch acceleration on fpga and its applications in network anomaly detection. *IEEE Transactions on Parallel and Distributed Systems*, 29(4):929–942, 2018.

[32] Jose Fernando Zazo, Sergio Lopez-Buedo, Mario Ruiz, and Gustavo Sutter. A single-fpga architecture for detecting heavy hitters in 100 gbit/s ethernet links. In *2017 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pages 1–6. IEEE, 2017.

[33] Da Tong and Viktor Prasanna. High throughput sketch based online heavy change detection on fpga. In *2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pages 1–8. IEEE, 2015.

[34] David Nguyen, Gokhan Memik, Seda Ogrenci Memik, and Alok Choudhary. Real-time feature extraction for high speed networks. In *International Conference on Field Programmable Logic and Applications, 2005.*, pages 438–443. IEEE, 2005.

[35] Antonio Saavedra, Cecilia Hernández, and Miguel Figueroa. Heavy-hitter detection using a hardware sketch with the countmin-cu algorithm. In *2018 21st Euromicro Conference on Digital System Design (DSD)*, pages 38–45. IEEE, 2018.

[36] Chi Wang and Bailu Ding. Fast approximation of empirical entropy via subsampling. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 658–667. ACM, 2019.

[37] Kuai Xu, Zhi-Li Zhang, and Supratik Bhattacharyya. Internet traffic behavior profiling for network security monitoring. *IEEE/ACM Transactions on Networking (TON)*, 16:1241–1252, 2008.

[38] Vyas Sekar, Michael K Reiter, and Hui Zhang. Revisiting the case for a minimalist approach for network flow monitoring. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 328–341, 2010.

[39] Theophilus Wellem, Yu-Kuen Lai, Chao-Yuan Huang, and Wen-Yaw Chung. A flexible sketch-based network traffic monitoring infrastructure. *IEEE Access*, 7:92476–92498, 2019.

[40] J Almeida, AZ Broder, P Cao, and L Fan. A scalable wide-area web cache sharing protocol. *SIGCOMM98*, 1998.

[41] Tian Bu, Jin Cao, Aiyou Chen, and Patrick PC Lee. A fast and compact method for unveiling significant patterns in high speed networks. In *IEEE INFOCOM 2007-26th IEEE International Conference on Computer Communications*, pages 1893–1901. IEEE, 2007.

[42] Aiyou Chen, Yu Jin, Jin Cao, and Li Erran Li. Tracking long duration flows in network traffic. In *2010 Proceedings IEEE INFOCOM*, pages 1–5. IEEE, 2010.

[43] Graham Cormode and Minos Garofalakis. Sketching streams through the net: Distributed approximate query tracking. In *Proceedings of the 31st international conference on Very large data bases*, pages 13–24. VLDB Endowment, 2005.

[44] Dhiman Barman, Piyush Satapathy, and Gianfranco Ciardo. Detecting attacks in routers using sketches. In *2007 Workshop on High Performance Switching and Routing*, pages 1–6. IEEE, 2007.

[45] Christian Callegari and North Cyprus. Statistical approaches for network anomaly detection. *Proc. ICIMP*, 2009.

[46] Graham Cormode and Marios Hadjieleftheriou. Finding frequent items in data streams. *Proceedings of the VLDB Endowment*, 1(2):1530–1541, 2008.

[47] Tong Yang, Yang Zhou, Hao Jin, Shigang Chen, and Xiaoming Li. Pyramid sketch: A sketch framework for frequency estimation of data streams. *Proceedings of the VLDB Endowment*, 10(11):1442–1453, 2017.

[48] Yin Zhang, Sumeet Singh, Subhabrata Sen, Nick Duffield, and Carsten Lund. Online identification of hierarchical heavy hitters: algorithms, evaluation, and applications. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 101–114. ACM, 2004.

[49] George Kollios, John W Byers, Jeffrey Considine, Marios Hadjieleftheriou, and Feifei Li. Robust aggregation in sensor networks. *IEEE Data Eng. Bull.*, 28(1):26–32, 2005.

[50] Tong Yang, Alex X Liu, Muhammad Shahzad, Yuankun Zhong, Qiaobin Fu, Zi Li, Gaogang Xie, and Xiaoming Li. A shifting bloom filter framework for set queries. *Proceedings of the VLDB Endowment*, 9(5):408–419, 2016.

[51] Tong Yang, Alex X Liu, Muhammad Shahzad, Dongsheng Yang, Qiaobin Fu, Gaogang Xie, and Xiaoming Li. A shifting framework for set queries. *IEEE/ACM Transactions on Networking*, 25(5):3116–3131, 2017.

[52] Qi George Zhao, Mitsunori Ogihara, Haixun Wang, and Jun Jim Xu. Finding global icebergs over distributed data sets. In *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 298–307. ACM, 2006.

[53] Graham Cormode, Theodore Johnson, Flip Korn, Shan Muthukrishnan, Oliver Spatscheck, and Divesh Srivastava. Holistic udafs at streaming speeds. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 35–46. ACM, 2004.

[54] Rob Pike, Sean Dorward, Robert Griesemer, and Sean Quinlan. Interpreting the data: Parallel analysis with sawzall. *Scientific Programming*, 13(4):277–298, 2005.

[55] Cisco Visual Networking Index. Forecast and methodology, 2015–2020. *White paper*, pages 1–41, 2016.

[56] Tong Yang, Gaogang Xie, YanBiao Li, Qiaobin Fu, Alex X Liu, Qi Li, and Laurent Mathy. Guarantee ip lookup performance with fib explosion. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 39–50. ACM, 2014.

[57] Tong Yang, Haowei Zhang, Hao Wang, Muhammad Shahzad, Xue Liu, Qin Xin, and Xiaoming Li. *FID-sketch: an accurate sketch to store frequencies in data streams. World Wide Web*, pages 1–22, 2018.

[58] Graham Cormode and Muthu Muthukrishnan. Approximating data with the count-min sketch. *IEEE software*, 29(1):64–69, 2011.

[59] Amit Goyal, Hal Daumé III, and Graham Cormode. Sketch algorithms for estimating point queries in nlp. In *Proceedings of the 2012 joint conference on empirical methods in natural language processing and computational natural language learning*, pages 1093–1103. Association for Computational Linguistics, 2012.

[60] Amit Goyal and Hal Daumé III. Approximate scalable bounded space sketch for large data nlp. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 250–261. Association for Computational Linguistics, 2011.

[61] Ran Ben Basat, Gil Einziger, Michael Mitzenmacher, and Shay Vargaftik. Faster and more accurate measurement through additive-error counters. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pages 1251–1260. IEEE, 2020.

[62] Qingjun Xiao, Zhiying Tang, and Shigang Chen. Universal online sketch for tracking heavy hitters and estimating moments of data streams. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pages 974–983. IEEE, 2020.

[63] A. Saavedra, H. Lehnert, C. Hernández, G. Carvajal, and M. Figueroa. Mining discriminative k-mers in dna sequences using sketches and hardware acceleration. *IEEE Access*, 8:114715–114732, 2020.

[64] Min Cai, Jianping Pan, Yu-Kwong Kwok, and Kai Hwang. Fast and accurate traffic matrix measurement using adaptive cardinality counting. *MineNet*, 5:205–206, 2005.

[65] Philippe Flajolet and G Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of computer and system sciences*, 31(2):182–209, 1985.

[66] Marianne Durand and Philippe Flajolet. Loglog counting of large cardinalities. In *European Symposium on Algorithms*, pages 605–617. Springer, 2003.

[67] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. In *Discrete Mathematics and Theoretical Computer Science*, pages 137–156. Discrete Mathematics and Theoretical Computer Science, 2007.

[68] Stefan Heule, Marc Nunkesser, and Alex Hall. Hyperloglog in practice: Algorithmic engineering of a state of the art cardinality estimation algorithm. In *Proceedings of the EDBT 2013 Conference*, Genoa, Italy, 2013.

[69] David D Lewis. Feature selection and feature extraction for text categorization. In *Proceedings of the workshop on Speech and Natural Language*, pages 212–217. Association for Computational Linguistics, 1992.

[70] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[71] Gavin Brown, Adam Pocock, Ming-Jie Zhao, and Mikel Luján. Conditional likelihood maximisation: a

unifying framework for information theoretic feature selection. *Journal of machine learning research*, 13(Jan):27–66, 2012.

[72] Austin Appleby. Smhasher & murmurhash (2012). *URL http://code. google. com/p/smhasher*, 2015.

[73] Petar D. Bojovic, Ilija Basicevic, Stanislav Očovaj, and Miroslav Popovic. DDoS attack scoreboard dataset, 2017. data retrieved from Mendeley, `https://data.mendeley.com/datasets/psjxnzsxyx/2`.

[74] CAIDA. The CAIDA UCSD Anonymized Internet Traces, 2008. data retrieved from CAIDA, `https://www.caida.org/data/passive/passive_dataset.xml`.

[75] Romain Fontugne, Pierre Borgnat, Patrice Abry, and Kensuke Fukuda. MAWILab: Combining Diverse Anomaly Detectors for Automated Anomaly Labeling and Performance Benchmarking. In *ACM CoNEXT '10*, Philadelphia, PA, December 2010.

[76] Damu Ding, Marco Savi, and Domenico Siracusa. Estimating logarithmic and exponential functions to track network traffic entropy in p4. In *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*, pages 1–9. IEEE, 2020.