UNIVERSIDAD DE CONCEPCIÓN
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

Algoritmos y aceleradores hardware para estimación de entropía de flujos en redes de datos

Tesis para optar al grado de Doctora en Ciencias de la Ingeniería
con mención en Ingeniería Eléctrica

Yaime Fernández Jiménez
Concepción - Chile
2025

Profesores Guía:
Dr. Miguel Figueroa Toro
Dra. Cecilia Hernández Rivas

Universidad de Concepción
Facultad de Ingeniería
Departamento de Ingeniería Eléctrica

Profesores Guía:
Dr. Miguel Figueroa T.
Dra. Cecilia Hernández R.

Algoritmos y aceleradores hardware para estimación de entropía de flujos en redes de datos

Yaime Fernández Jiménez

Informe de tesis
Doctorado en Ciencias de la Ingeniería con mención en Ingeniería Eléctrica

25 de agosto de 2025

Resumen

La entropía empírica de Shannon se emplea en la monitorización y clasificación del tráfico de red, con aplicaciones en la detección de anomalías y la gestión de recursos. En enlaces de red modernos, el cálculo de la entropía debe considerar que los datos se generan en *streaming*, de forma continua a altas velocidades. Además, en muchas aplicaciones los datos recientes adquieren mayor relevancia. Por ello, se utiliza el modelo de ventana deslizante que mantiene actualizadas las mediciones y refleja tendencias actuales del tráfico, pero implica el desafío adicional de eliminar de forma continua los datos obsoletos. Los aceleradores hardware basados en FPGA pueden alcanzar un alto rendimiento en estos escenarios a altas velocidades, explotando el paralelismo inherente del algoritmo con baja latencia y consumo energético. Sin embargo, estas plataformas tienen recursos de memoria interna limitados. Por lo que se requiere el diseño de algoritmos de estimación de entropía que puedan ser mapeados eficientemente a arquitecturas hardware, con bajo uso de memoria y alto paralelismo para maximizar su rendimiento.

En este trabajo se proponen dos algoritmos para estimar la entropía empírica de Shannon de flujos de red, utilizando las frecuencias de los top-K, y asumiendo una distribución estadística para el resto. El primero realiza la estimación en intervalos de tiempo discreto y el segundo en ventanas de tiempo deslizantes. Ambos algoritmos fueron implementados en hardware utilizando *sketches* para la estimación de frecuencias y cardinalidad. Los *sketches* son estructuras de datos probabilísticas que implementan algoritmos de *streaming* que permiten operaciones en línea, ofrecen un alto grado de paralelismo y hacen un uso eficiente del espacio disponible. Además, se implementaron arreglos de colas de prioridad aproximadas para almacenar los flujos de mayor frecuencia; estas estructuras son altamente paralelizables y eficientes en el uso de memoria.

Ambos algoritmos se evaluaron en trazas de red reales. El primer algoritmo muestra en intervalos con hasta 50 millones de flujos, un error relativo promedio de 0,69 % en la estimación de la entropía empírica. El algoritmo utiliza más de tres órdenes de magnitud menos de memoria que un método exacto de cálculo de entropía. El acelerador hardware implementado en un FPGA Virtex UltraScale+ XCU280 opera a una velocidad de línea de más de 200 Gbps y una latencia de estimación de 16 μs. Los resultados del segundo algoritmo, en ventanas de 60s y deslizamiento 6s, con hasta 2 millones de flujos por ventana, muestran errores relativos promedio menores al 0.45 % con una desviación estándar de 0.14 %. El algoritmo utiliza la mitad de la memoria y alcanza errores menores que replicar el primer algoritmo en cada ventana. El acelerador hardware implementado en un FPGA Virtex XCU55 UltraScale+ opera a una velocidad de línea de más de 158 Gbps.

"*Lo que comienza como interés se convierte en pasión a través de la perseverancia a largo plazo.*"

-A. Duckworth

Durante las diferentes etapas de este trabajo he recibido el apoyo financiero del Programa de Becas para Estudios de Doctorado Nacional de la Agencia Nacional de Investigación y Desarrollo de Chile (ANID) del Gobierno de Chile.

Tabla de contenidos

Lista de figuras VII

Lista de tablas IX

Lista de algoritmos X

Lista de acrónimos XI

Capítulo 1 Introducción 1

1.1 Introducción general . . . 1
1.2 Hipótesis . . . 4
1.3 Objetivo general . . . 4
1.4 Objetivos específicos . . . 4
1.5 Temario . . . 5

Capítulo 2 Medición de entropía en redes de datos 7

2.1 Introducción . . . 7
2.2 Mediciones de tráfico de red . . . 7
2.3 Medición de entropía en redes . . . 8
2.3.1 Mediciones de entropía en un intervalo discreto . . . 9
2.3.2 Mediciones de entropía en ventanas deslizantes . . . 12
2.4 Aceleración hardware . . . 14
2.4.1 Aceleración hardware usando P4 . . . 14
2.4.2 Aceleración hardware basada en FPGA . . . 17
2.5 Discusión . . . 20

Capítulo 3 Estructuras de datos 21

3.1 Introducción . . . 21
3.2 Funciones *hash* . . . 21
3.3 Estructuras probabilísticas: *sketches* . . . 22
3.3.1 Estimación de cardinalidad . . . 22
3.3.2 Estimación de frecuencias . . . 24
3.4 Arreglo de colas de prioridad para detección de top-K . . . 25
3.5 Discusión . . . 26

Capítulo 4 Entropía en tiempo discreto **27**
4.1 Introducción . . . 27
4.2 Entropía empírica de Shannon . . . 27
4.3 Método . . . 28
4.4 Algoritmo . . . 31
4.5 Arquitectura en FPGA . . . 36
4.5.1 Módulo de procesamiento de paquetes en tiempo discreto . . . 36
4.5.2 Módulo de estimación de entropía . . . 37
4.6 Resultados . . . 40
4.6.1 Parámetros del algoritmo . . . 41
4.6.2 Resultados del algoritmo . . . 45
4.6.3 Rendimiento del acelerador . . . 53
4.7 Discusión . . . 56

Capítulo 5 Entropía en ventanas de tiempo deslizantes **57**
5.1 Introducción . . . 57
5.2 Modelo de ventanas deslizantes . . . 57
5.3 Método . . . 58
5.4 Algoritmo de estimación de entropía en ventanas de tiempo deslizantes . . . 60
5.5 Arquitectura en FPGA . . . 64
5.5.1 Módulo de procesamiento de paquetes en ventanas de tiempo deslizantes 65
5.5.2 Módulo de estimación de entropía . . . 67
5.6 Resultados . . . 69
5.6.1 Parámetros del algoritmo . . . 70
5.6.2 Resultados del algoritmo . . . 77
5.6.3 Rendimiento del acelerador . . . 79
5.7 Discusión . . . 81

Capítulo 6 Conclusiones y trabajo futuro **82**

Anexo A Publicaciones **84**

Referencias bibliográficas **94**

Lista de figuras

3.1 Estructura del *sketch* de cardinalidad HyperLogLog. 23
3.2 Estructura genérica de *sketches* de cuentas o de estimación de frecuencias. . . . 24
3.3 Estructura del arreglo de colas de prioridad. 26

4.1 Histograma normalizado de probabilidades de los flujos para la traza Mawi-20181400 en escala log-log. La curva roja representa las frecuencias normalizadas utilizando una escala lineal y ordenadas por magnitud, lo que facilita el análisis de las propiedades estadísticas que sustentan el método. La línea discontinua corresponde a un ajuste de ley de potencias del histograma. Esta ley de potencias se ajusta aproximadamente a los datos tanto en la región de frecuencias altas como bajas, lo que permite estimar los parámetros de la ley de potencias empleando únicamente los elementos más frecuentes. 29
4.2 Arquitectura del acelerador hardware para la estimación de entropía en intervalos de tiempo discreto. 36
4.3 Módulo de cálculo del logaritmo en base 2. 38
4.4 Ejecución del algoritmo de ordenamiento *radix-sort* en el acelerador. 39
4.5 Módulo de estimación de $\hat{\alpha}$. 39
4.6 Módulo de cálculo de potencias. 40
4.7 Error de estimación de entropía del método propuesto relativo al valor exacto de la entropía calculada con la Ecuación (4.4), en función de K. 42
4.8 Error de estimación de entropía del algoritmo relativo a la entropía estimada mediante la Ecuación. (4.10), en función del parámetro de precisión p del HLL. 43
4.9 Histogramas normalizados de probabilidades de los flujos, en escala log-log, para seis trazas. Cada gráfico compara los datos reales de la traza con las estimaciones producidas por el algoritmo propuesto. Los puntos entre 1 y K corresponden a frecuencias estimadas por el Count Sketch y capturadas por el PQA. Para valores entre K y $\hat{L}$, el algoritmo estima las frecuencias mediante un ajuste de ley de potencias, mientras que para valores superiores a $\hat{L}$, las frecuencias se aproximan como un paquete por flujo. 49

5.1 Modelo de ventana deslizante utilizado en la estimación de entropía empírica. . . 59
5.2 Esquema en bloques de la estructura cíclica formada por once *sketches* CS utilizada para la estimación de frecuencia en ventanas de tiempo deslizantes. 60

5.3 Esquema en bloques de la estructura cíclica formada por once HLL utilizada para la estimación de cardinalidad en ventanas de tiempo deslizantes. 61
5.4 Arquitectura del acelerador hardware para la estimación de entropía en ventanas de tiempo deslizantes. 65
5.5 Módulo de estimación de frecuencias usando la unión de CSs. 66
5.6 Errores promedio de estimación de la entropía empírica del método propuesto, relativo al valor exacto de la entropía calculada con la Ecuación (4.4), en función de K. 71
5.7 Errores promedio de estimación de entropía del algoritmo relativo a la entropía estimada mediante la Ecuación (4.10), en función del parámetro de precisión p de los *sketches* HLL. 72
5.8 Entropía exacta calculada con la Ecuación (4.4), junto con las estimaciones del método y el algoritmo propuesto, para seis trazas de la Tabla 5.1. 78

Lista de tablas

4.1 Trazas de tráfico de red utilizadas en los experimentos, donde M, N y H_{norm} representan el número total de paquetes, la cardinalidad de los flujos y la entropía normalizada exacta, respectivamente. 41
4.2 Error de estimación del algoritmo para dos *sketches* de cuentas: CS y CM-CU y seis dimensiones diferentes. 44
4.3 Error de estimación de entropía del algoritmo en función de las dimensiones del PQA, relativo a la estimación del método de la Ecuación (4.10). 46
4.4 Entropía exacta calculada con la Ecuación (4.4), junto con las estimaciones del método y el algoritmo propuesto, para las 16 trazas de la Tabla 4.1. 47
4.5 Errores de estimación absolutos y relativos obtenidos mediante la Ecuación (4.10) y el Algoritmo 1, usando la entropía exacta como referencia. 48
4.6 Parámetros utilizados en la implementación actual de algoritmos publicados previamente y del algoritmo propuesto. 50
4.7 Errores relativos de estimación de entropía obtenidos por algoritmos publicados previamente y el algoritmo propuesto, utilizando como referencia la entropía exacta y normalizada de los flujos. 51
4.8 Uso máximo de memoria del algoritmos que calcula la entropía exacta de los flujos, métodos de estimación anteriores y el método propuesto en este trabajo. . 52
4.9 Latencia de las implementaciones en C++ de los algoritmos de estimación de entropía propuestos previamente y el método presentado en este trabajo. 53
4.10 Errores de estimación absolutos y relativos obtenidos por el Algoritmo 1 y el acelerador, utilizando la entropía exacta como referencia. 55
4.11 Utilización de recursos del FPGA por núcleo (kernel). 55

5.1 Trazas de MawiLab utilizadas en los experimentos, donde $\bar{M}$, $\bar{N}$ y $\bar{H}$ representan los valores promedio del número total de paquetes, la cardinalidad de los flujos y la entropía normalizada exacta, respectivamente; y σ_M, σ_N y σ_H las desviaciones estándar correspondientes. El número de ventanas de cada traza es representado por W_k. 70
5.2 Errores promedio en la estimación de cardinalidad de los algoritmo SHLL y HLL usando $p = 2$ y $p = 8$, respectivamente; relativo a la cardinalidad exacta N. . . . 73

5.3 Errores promedio en la estimación de la entropía, relativos a la entropía estimada mediante la Ecuación (4.10), usando las frecuencias estimadas por la unión de diez CS. 74
5.4 Errores promedio en la estimación de la entropía respecto a la entropía estimada mediante la Ecuación (4.10), usando las frecuencias estimadas por la unión de diez CS con $w = 16384$, $d = 9$ y saturación de contadores. 74
5.5 Errores promedio en la estimación de la entropía respecto a la entropía estimada mediante la Ecuación (4.10), usando las frecuencias estimadas por un CS en cada ventana W. 75
5.6 Errores promedio en la estimación de las frecuencias top-K utilizando UC *Sketch* y la unión de diez CS, relativos a los valores reales de frecuencias. 76
5.7 Errores promedio en la estimación de la entropía, respecto a la entropía estimada mediante la Ecuación (4.10), usando diferentes dimensiones del PQA. 77
5.8 Errores promedios absolutos y relativos de estimación obtenidos mediante la Ecuación (4.10) y el Algoritmo 2 y 3, usando la entropía exacta como referencia. 79
5.9 Utilización de recursos del FPGA por núcleo (kernel). 80

Lista de algoritmos

1 Estimación de la entropía empírica normalizada en intervalos de tiempo discretos. 32

2 Procesamiento de los paquetes de red en ventanas de tiempo deslizantes. 63

3 Estimación de la entropía empírica normalizada en ventanas de tiempo deslizantes. 64

Lista de acrónimos

AMS Alon-Matias-Szegedy.

ASIC Application-Specific Integrated Circuit.

AXI Advanced eXtensible Interface.

BRAM Block Random Access Memory.

CAIDA Center of Applied Internet Data Analysis.

CM Count Min.

CM-CU Count Min with Conservative Update.

CS Count Sketch.

CUSUM Cumulative Sum Control Chart.

DDoS Distributed Denial of Service.

DDR3 Double Data Rate 3.

DSL Domain-Specific Language.

DRAM Dynamic Random-Access Memory.

ECM Exponential Count-Min.

FPGA Field-Programmable Gate Array.

HLL HyperLogLog.

HSS Half-Sliding Sketch.

IP Internet Protocol.

LMB Local Memory Bus.

LPM Longest Prefix Match.

LSM Least Squares Method.

LSTM-RNN Long Short-Term Memory Recurrent Neural Network.

LUT Lookup Table.

P4 Programming Protocol-independent Packet Processors.

PQA Priority Queue Array.

QDRII Quad Data Rate II.

RTL Register-Transfer Level.

SHLL Sliding HyperLogLog.

Smart NIC Smart Network Interface Card.

SRAM Static Random Access Memory.

SWAMP Sliding Window Approximate Measurement Protocol.

TCAM Ternary Content-Addressable Memory.

TCP Transmission Control Protocol.

TDP Thermal Design Power.

UDP User Datagram Protocol.

Capítulo 1. Introducción

1.1. Introducción general

La entropía es una medida de la teoría de la información que cuantifica la incertidumbre asociada a los posibles resultados de una variable estocástica [1]. Es utilizada ampliamente en aplicaciones como la monitorización del tráfico de red [2, 3], la detección de anomalías en datos biomédicos [4] y el análisis del impacto de la política monetaria en redes financieras [5]. En lugar de estimar la entropía de una distribución teórica, la mayoría de las aplicaciones calculan la entropía empírica, que se obtiene directamente a partir de la frecuencia de aparición de los valores de la variable en un conjunto de datos.

En la monitorización de tráfico de red, los datos se representan como un conjunto de flujos o secuencias de paquetes observados durante un intervalo de tiempo determinado, con atributos comunes como direcciones IP de origen y destino, protocolos de capa 4 como el Protocolo de Control de Transmisión (*Transmission Control Protocol*, TCP) o el Protocolo de Datagrama de Usuario (*User Datagram Protocol*, UDP) y puertos de origen y destino. En este contexto, un valor alto de entropía indica diversidad y aleatoriedad en el tráfico, ejemplo cuando participan varias direcciones IP, puertos o protocolos diferentes. Mientras que un valor bajo implica que la distribución está concentrada, con ciertos elementos dominando el tráfico; por ejemplo, la mayoría de los paquetes provienen o se dirigen a una única dirección IP. La entropía es utilizada en la detección de anomalías [6, 7, 8] y la clasificación del tráfico [9, 10]. En la detección de anomalías permite identificar eventos anómalos que pueden no reflejarse en cambios en el volumen de tráfico. En clasificación, la entropía mide la distancia entre grupos o *clusters* [11, 9].

El cálculo de la entropía empírica en grandes conjuntos de datos puede ser computacionalmente costoso, ya que requiere mantener un contador para cada elemento distinto con el fin de calcular su frecuencia. En el contexto del tráfico de red, este cálculo presenta desafíos adicionales. En primer lugar, los identificadores de flujo combinan múltiples atributos de los paquetes, lo que da como resultado posible un gran número de elementos distintos en el conjunto. En segundo lugar, en redes modernas, las secuencias de paquetes se reciben en *streaming*. El término *streaming* de datos se refiere a la situación en la que los datos se generan de forma continua a alta velocidad, superiores a 100 Gbps y deben ser procesados en tiempo real para facilitar el análisis de datos y la toma de decisiones [12]. En tercer lugar, el rápido desarrollo de Internet impulsa a las aplicaciones a centrarse en las estadísticas recientes de los flujos, la importancia

de la información disminuye con el tiempo [13]. Por lo tanto, es utilizado el modelo de ventana deslizante para obtener actualizaciones constantes de las mediciones de los últimos N elementos o de los que llegan en las últimas N unidades de tiempo [14]. En este último caso, basado en el tiempo se puede reflejar la tendencia general de la red sin afectaciones por la cantidad de elementos medidos. Aunque el modelo de ventana deslizante limita la cantidad de datos a procesar, tiene el desafío de eliminar continuamente datos antiguos [15]. La eliminación adecuada requiere el almacenamiento de los identificadores de flujo pero es ineficiente, dado que los dispositivos de red, como *routers* y *switches*, tienen recursos limitados de memoria y procesamiento [16].

Ante el desafío de procesar grandes volúmenes de datos a velocidades de línea son utilizados *switches* programables o aceleradores implementados en hardware dedicado. Los *switches* programables modernos admiten el lenguaje P4: *Programming Protocol-independent Packet Processors* y pueden procesar hasta miles de millones de paquetes por segundo en el plano de datos con retardos inferiores a microsegundos [17]. El lenguaje P4 ofrece ventajas como reconfigurabilidad, independencia de protocolo e independencia de plataforma. No obstante, presenta limitaciones importantes como la falta de soporte para bucles, aritmética de punto flotante y operaciones fundamentales como división, logaritmos y funciones exponenciales [18]. En otros casos, como alternativa o complemento se utilizan aceleradores en hardware dedicado, implementados en Circuitos Integrados de Aplicación Específica (*Application Specific Integrated Circuits*, ASIC) o en Arreglos de Compuertas Lógicas Programables (*Field Programmable Gate Array*, FPGA). Estos pueden procesar un *stream* de datos a velocidades de línea, explotando el paralelismo del algoritmo con baja latencia y consumo de energía [19, 20, 21].

Una limitación común a los *switches* programables y a los aceleradores en hardware dedicado es la cantidad de memoria disponible en el dispositivo. La memoria *on-chip* ofrece mayor ancho de banda agregado y paralelismo que la memoria externa, por lo que los algoritmos que maximizan el uso de la memoria interna pueden lograr un mejor rendimiento en estas plataformas [22]. En particular, los algoritmos de *streaming* basados en *sketches* se han utilizado para estimar propiedades del tráfico de red con alta precisión y bajo uso de memoria [23, 24, 25]. Los *sketches* son estructuras de datos probabilísticas que soportan operaciones en línea y utilizan memoria sublineal. 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 [19]. Sin embargo, los *sketches* proveen aproximaciones con errores de precisión acotados, lo que comúnmente es tan útil como los resultados exactos [26]. Otra ventaja destacada de los *sketches* es su capacidad de combinación o unión, lo que facilita el procesamiento distribuido y el análisis conjunto de información proveniente de múltiples dispositivos en la red. Además, su alto grado de para-

lelismo los hace especialmente adecuados para su implementación en hardware. Ejemplos del uso de *sketches* en la medición de tráfico de red son *Count Sketch* (CS) [27] y *Count Min with Conservative Update* (CM-CU) [28] que pueden estimar el tamaño de los flujos, mientras que *HyperLogLog* (HLL) y *Supersketch* [29, 30] estiman la cardinalidad, es decir, la cantidad de flujos distintos. En [25], los autores presentaron una revisión de las aplicaciones de los *sketches* en la medición de tráfico de red. Un estudio posterior [13] amplía esta revisión, enfocándose en *sketches* diseñados para medición del tráfico usando el modelo de ventanas deslizantes, consideró sus estructuras, operaciones soportadas, tareas de medición, plataformas de implementación y rendimiento. Los *sketches* en ventanas deslizantes, se caracterizan generalmente por un mayor uso de memoria debido a la necesidad de almacenar marcas de tiempo, en algunos casos, y a la complejidad de gestionar entradas obsoletas.

En [31], se propuso un algoritmo y un acelerador hardware basado en FPGA para estimar la entropía empírica de los flujos identificados por la misma dirección IP de origen. El algoritmo calculó la entropía considerando únicamente los K flujos más frecuentes y utilizó un *sketch* para estimar las frecuencias de los flujos y una cola de prioridad para almacenar las 10,240 frecuencias más altas. El acelerador operó a una velocidad de 181 Gbps, pero el error en la estimación de entropía alcanzó hasta un 14 %, dependiendo de la traza de datos [32, 33]. En [34], mejoramos el algoritmo asumiendo que los flujos fuera de los top-K tienen todos la misma frecuencia. Esta mejora añade un *sketch* de cardinalidad para estimar el número total de flujos y redujo el error medio al 1.65 %, con un máximo del 3 %. También permitió reducir el tamaño de la cola de prioridad a 8,192 elementos e incrementar la velocidad de línea a 204 Gbps. Sin embargo, al utilizar identificadores de flujo más complejos, por ejemplo, el conjunto de cinco atributos compuesto por las direcciones y puertos de origen/destino y el protocolo, el número de flujos aumenta drásticamente. Bajo estas condiciones, la suposición de frecuencia uniforme para los flujos no medidos degrada significativamente la precisión de la estimación, haciendo que el método no sea adecuado para detectar de forma fiable anomalías críticas como los ataques de Denegación de Servicio Distribuida (DDoS) [35, 11]. Además, este algoritmo solo opera en un intervalo de observación fijo, igual a la duración de la traza.

En esta tesis se presenta el diseño de dos algoritmos de *streaming* basados en *sketches* y sus aceleradores hardware para la estimación de la entropía empírica de Shannon de flujos de red. El identificador de flujo utilizado se compone de las direcciones IP y puertos de origen/destino, y el protocolo. Los algoritmos utilizan las fecuencias de los flujos más frecuentes o top-K para estimar la frecuencia del resto, asumiendo que estas siguen una distribución de ley de potencias. El primer algoritmo estima la entropía empírica en intervalos de tiempo discreto y el segundo

en ventanas de tiempo deslizantes. Ambos algoritmos usan *sketches* para estimar la frecuencia y cardinalidad de los flujos y un arreglo de colas de prioridad para almacenar las frecuencias top-K. Además, se diseñan arquitecturas hardware que pueden ejecutar eficientemente los algoritmos en FPGA.

1.2. Hipótesis

La capacidad de resumir grandes volúmenes de datos en un espacio sublineal con errores de precisión acotados asociada a los algoritmos basados en *sketches* puede utilizarse en la aceleración hardware de mediciones de tráfico de red, aplicables a la detección de anomalías; con una alta tasa de procesamiento en tiempo real y bajo consumo de potencia.

1.3. Objetivo general

Diseñar arquitecturas de hardware que provean aceleración a estimaciones de entropía empírica de Shannon, aplicable a la detección de anomalías, con una alta tasa de procesamiento en tiempo real y bajo consumo de potencia usando algoritmos basados en *sketches*.

1.4. Objetivos específicos

1. Proponer un método matemático para la estimación de la entropía empírica de Shannon, como métrica para la detección de anomalías.
2. Caracterizar los *sketches* utilizados en mediciones de tráfico de red para la estimación de entropía empírica de Shannon.
3. Analizar los requerimientos de diseño en hardware dedicado que presenta el uso de algoritmos basados en *sketches* en mediciones de tráfico de red para la estimación de entropía empírica de Shannon.
4. Implementar en hardware dedicado una arquitectura de medición de tráfico de red para la estimación de entropía empírica de Shannon usando *sketches*.

5. Adaptar el algoritmo y la arquitectura propuesta para la estimación de la entropía empírica de Shannon para su uso en ventanas de tiempo deslizantes.

6. Validar el funcionamiento de las arquitecturas diseñadas.

1.5. Temario

- En el Capítulo 2 se presenta el estado del arte relacionado con los algoritmos de *streaming* para la medición de la entropía empírica de Shannon en el tráfico de red. Se analizan algoritmos que emplean ventanas de observación fijas y el modelo de ventanas deslizantes. Además, se revisan algoritmos basados en *sketches* diseñados para la estimación de frecuencia, cardinalidad y detección de flujos top-K. Finalmente, se describen implementaciones de estos algoritmos tanto en *switches* programables como en aceleradores hardware, destacando sus ventajas, limitaciones y rendimiento en entornos de red de alta velocidad.

- En el Capítulo 3 se describe el marco teórico de la tesis. Se analiza el funcionamiento de las estructuras de datos basadas en *sketches* para estimaciones de frecuencia y cardinalidad, utilizadas en el diseño de algoritmos y aceleradores hardware que estiman la entropía empírica. Además se analiza un arreglo de colas de prioridad que almacena las frecuencias estimadas de los flujos top-K y la función *hash* empleada para distribuir uniformemente lo elementos de entrada o flujos en dichas estructuras.

- En el Capítulo 4 se propone un método y algoritmo de *streaming* para la estimación de entropía empírica de los flujos de red en un intervalo de tiempo discreto y su acelerador hardware. El método uiliza las fecuencias de los flujos top-K para estimar la frecuencia del resto de la distribución, asumiendo que estas siguen una ley de potencias. El algoritmo usa *sketches* para estimar la cardinalidad y frecuencias de los flujos; y un arreglo de colas de prioridad para almacenar las frecuencias de los top-K. Además, se diseña la arquitectura de un acelerador hardware para el algoritmo propuesto. Finalmente, se presentan los resultados obtenidos por el algoritmo y el acelerador en conjuntos de trazas reales.

- En el Capítulo 5 se adapta el algoritmo de *streaming* basado en *sketches* propuesto en el Capitulo 4 para su uso en ventanas de tiempo deslizantes. El algoritmo estima la entropía empírica en cada ventana de tiempo de 60s y deslizamiento 6s, utilizando once *sketches* de cardinalidad, once *sketches* de cuentas y once arreglos de colas de prioridad para almacenar las frecuencias top-K en cada ventana. Las estimaciones de *sketches* de frecuencia no

deslizantes en ventanas de tiempo de tamaño s se combinan para determinar las frecuencias de los flujos en cada ventana, buscando un buen compromiso entre precisión y uso de memoria. Además, se presenta el diseño de la arquitectura de un acelerador hardware para el algoritmo adaptado. Finalmente, se presentan los resultados obtenidos por el algoritmo en trazas reales y el rendimiento del acelerador.

- En el Capítulo 6 se presentan las conclusiones del trabajo desarrollado y el trabajo futuro.

Capítulo 2. Medición de entropía en redes de datos

2.1. Introducción

En el presente capítulo se establecen los fundamentos teóricos de la problemática de investigación. Se presentan las principales mediciones de tráfico de red, incluida la entropía empírica de Shannon y su importancia en aplicaciones de monitoreo. Se analizan los desafíos de la estimación de entropía en redes, los principales algoritmos y métodos que utilizan. Posteriormente, se analizan implementaciones de estos algoritmos en hardware dedicado y en *switches* programables. Finalmente se discuten los principales aspectos del capítulo.

2.2. Mediciones de tráfico de red

La medición de red se refiere al proceso de recopilar y analizar estádisticas de tráfico para caracterizar su estado [13]. Se utilizan en aplicaciones de monitoreo, tales como la detección de anomalías [8], la clasificación de tráfico [10], el control de congestión [36] y el balanceo de carga [37]. Las mediciones de tráfico de red comunes son: la estimación del tamaño del flujo, la estimación de cardinalidad, la detección de anomalías de cambios o *heavy changes* e identificación de propagadores persistentes [25]. La medición del tamaño del flujo cuenta el número de *bytes* o paquetes para cada flujo activo durante un período de tiempo. Es aplicada en la detección de elementos frecuentes y la detección de *heavy hitters*, flujos con un volumen de tráfico anómalamente alto [38, 23]. La medición de cardinalidad se clasifica en la cardinalidad del *host* o del flujo. Esta última, determina el número flujos distintos en un *stream* de datos, también denominada dispersión [39, 40]. La cardinalidad del *host* o grado de conexión representa el número de pares distintos con los que este se comunica [41]; y puede dividirse en cardinalidad de destino, cardinalidad del puerto de destino y cardinalidad de la fuente. La medición o detección de anomalías de cambio o *heavy changes*, deriva un modelo de comportamiento normal basado en el historial del tráfico y busca cambios significativos que no coinciden con el modelo en un período de tiempo determinado [42]. Los cambios significativos en los patrones de tráfico; por ejemplo, el tamaño del flujo y la cardinalidad están relacionados con eventos anómalos como la distribución de *spam* [25]. La medición de propagadores persistente se refiere al número flujos distintos persistentes durante un número predefinido de períodos de tiempo consecutivos [43]. Esta medición es utilizado en la detección de *spam* y ataques de DDoS que utilizan una baja

tasa de paquetes durante un período de tiempo prolongado [44].

Otras mediciones derivadas de las básicas incluyen la de estimación de cuantiles, consulta de pertenencia para verificar si un elemento está en el intervalo de observación, la consulta de los top-K o las K frecuencias más altas y la entropía. En especial, la entropía es una medida de incertidumbre asociada a una variable aleatoria. Cuanto más aleatorios sean los valores en la distribución observada, mayor será su entropía. Por lo tanto, los cambios en la entropía de la dirección IP de origen, la dirección IP de destino o el puerto de destino que identifican a determinados flujos, pueden señalar eventos inusuales en la red. Su uso en el análisis del tráfico de red ofrece dos beneficios importantes: en primer lugar, aumenta la sensibilidad en la detección de eventos anómalos que pueden no manifestarse como cambios bruscos en el volumen de tráfico. En segundo lugar, entropía de diferentes atributos de tráfico proporciona información de diagnóstico adicional sobre la naturaleza de eventos anómalos, por ejemplo, permite distinguir entre gusanos, ataques de DDoS y escaneos, información que no está disponible en los enfoques de detección basados únicamente en el volumen de tráfico [45, 11]. Aunque generalizaciones de la entropía, como las formulaciones de Tsallis [46] y Rényi [47], han mostrado ventajas en escenarios con bajo volumen de tráfico [48, 49, 3], la entropía de Shannon sigue siendo ampliamente utilizada en los algoritmos actuales de análisis de tráfico y detección de anomalías [11, 50, 51, 52, 53, 18].

2.3. Medición de entropía en redes

La entropía empírica de Shannon se calcula directamente a partir de la frecuencia de aparición de los elementos en un conjunto de datos. Para su estimación se requiere conocer tanto el número de elementos distintos como la frecuencia de ocurrencia de cada uno. En redes modernas, el número de elementos distintos puede ser elevado, dado que los identificadores de flujo combinan múltiples atributos de los paquetes. Además, en estas redes las secuencias de paquetes se generan en *streaming*, es decir, de forma continua, a velocidad altas y con posibles cambios en el tiempo. Estas condiciones imponen restricciones significativas para el almacenamiento y procesamiento en tiempo real.

Algoritmos de medición de entropía limitan la cantidad de datos a procesar; utilizando modelos basados en ventanas físicas o lógicas. La ventana física se define por rangos de tiempos, por lo que su tamaño o número de elementos no se puede conocer a priori. La ventana lógica se define en términos del número de elementos. Dos modelos comunes en el análisis de tráfico de red, son el modelo de ventana fija o intervalo discreto y el modelo de ventana deslizante.

En el modelo de ventana fija, estas no se superponen y los datos se conservan si estan dentro de la ventana actual. En la ventana deslizante, los datos recientes son más importantes que los antiguos. Los límites de la ventana cambian con el tiempo, cada vez que se añaden elementos se eliminan antiguos.

Debido a que dispositivos de red, como *routers* y *switches*, tienen memoria y recursos de procesamiento limitados, realizar mediciones en *streaming* en una ventana de observación fija o deslizante es un desafío en escenarios reales [54]. Por lo tanto, se utilizan métodos aproximados para medir el tráfico de red; como los basados en muestreo y en *sketches*. El muestreo selecciona un subconjunto del conjunto representativo de los flujos de datos originales e infiere las características de estos. La cantidad de datos muestreados está relacionada con los datos originales y la frecuencia de muestreo. Una frecuencia de muestreo alta puede garantizar una alta precisión, pero implica una sobrecarga de memoria. Por el contrario, una frecuencia de muestreo baja disminuye el uso de memoria y la precisión. Los *sketches* son una familia de estructuras de datos probabilisticas diseñadas para estimar varias estadísticas de un *streaming* de datos. Brindan errores de precisión acotados y hacen uso sublineal de la memoria con respecto a los datos de entrada. Además, son adaptables para cooperar con otras técnicas de medición. Se puede usar en combinación con otros métodos estadísticos como el algoritmo *Cumulative Sum Control Chart* (CUSUM) [55], utilizado para monitorear cambios en el tráfico de red. En las siguientes secciones se analizan algoritmos de estimación de entropía basados en muestreo y en *sketches* para ventanas de observación fija y posteriormente para ventanas deslizantes.

2.3.1. Mediciones de entropía en un intervalo discreto

Los algoritmos de estimación de entropía en el tráfico de red propuestos en la literatura [11, 12, 52, 34] se clasifican en dos grupos principales [56]: basados en muestreo y basados en *sketches*. Lall *et al.* [11] propusieron dos algoritmos para la estimación de la entropía de Shannon, y demostraron que cambios en la entropía de la distribución de paquetes observados en diferentes puertos pueden indicar ataques de escaneo. El primer algoritmo estima el segundo momento de frecuencia utilizando el muestreo de *Alon-Matias-Szegedy* (AMS) [57]. El algoritmo ofrece sólidas garantías de precisión y uso de recursos. Las garantías teóricas son independientes de la distribuciones de tráfico subyacentes; provocando una diferencia de al menos un orden de magnitud menor entre las garantías de error de precisión teóricas y los errores observados en la práctica. El segundo algoritmo mejora el error de estimación al considerar los flujos más y menos frecuentes por separado. Los flujos muestreados más de una vez utilizan contadores exactos para

estimar su frecuencia y el resto se estima usando el primer algoritmo. El algoritmo logró alta precisión y alta velocidad de procesamiento, pero no estima la entropía en tiempo real [58].

En [52], los autores utilizaron submuestreo aleatorio para aproximar la entropía empírica de los elementos de un *stream*. Derivaron límites probabilísticos para el error de estimación y lo utilizaron para construir intervalos de confianza de la entropía empírica. Además, propusieron algoritmos para acelerar las aplicaciones de clasificación y filtrado, que muestrean progresivamente los datos hasta que pueden devolver la respuesta correcta con la probabilidad especificada. La evaluación de los algoritmos en conjuntos de datos reales demostraron una aceleración de hasta tres órdenes de magnitud con respecto a los métodos exactos, con un error prácticamente nulo. Posteriormente, en [59] los autores mejoraron la complejidad temporal, sus experimentos en conjuntos de datos reales mostraron mejoras en un orden de magnitud con la misma precisión.

Una alternativa para obtener mediciones aproximadas de la entropía empírica de los elementos en un *stream* es el uso de *sketches*. Clifford y Cosma [12] propusieron un *sketch* simple e insesgado basado en proyecciones lineales aleatorias [60], extraídas de una distribución α-estable con sesgo máximo, para la estimación de la entropía de Shannon. Los autores transformaron en línea distintos elementos del *stream* en realizaciones distintas de una variable estable de índice α y almacenaron combinaciones lineales ponderadas de estas realizaciones, replicadas independientemente k veces. Estas combinaciones lineales ponderadas o proyecciones aleatorias, forman el *sketch*, donde k se determina por la precisión deseada en la aproximación de la entropía. Estas propiedades permitieron estimar la entropía a partir del *sketch* asociado como el logaritmo de un promedio simple. Sus resultados mostraron una complejidad espacial casi óptima.

Callegari *et al.* [49] propusieron un sistema de detección de intrusos basado en variaciones de la entropía de distintos atributos del tráfico. Los autores utilizaron un *sketch* reversible tridimensional, que permite rastrear las direcciones IP responsables de las anomalías. El *sketch* utiliza dos conjuntos de funciones *hash* independientes, uno responsable de la construcción del *sketch* y el segundo para asignar cada elemento de entrada a un histograma, almacenados en los *buckets* de la tercera dimensión. Para la detección de anomalías, calcularon la divergencia de la entropía asociadas a cada *bucket* entre el *sketch* actual y uno de referencia. Si la divergencia entre los *sketches* excede un umbral predeterminado se detecta una anomalía. En trazas de tráfico real de MawiLab [33] utilizando la entropía de Shannon, sus resultados mostraron una tasa de detección superior al 85 % con una tasa de falsas alarmas del 15 %.

Frameworks recientes como *UnivMon* [61], *ElasticSketch* [62] y *LightGuardian* [63] utilizan *sketches* para múltiples mediciones del tráfico de red, incluida la identificación de flujos top-K y

la estimación de frecuencias y entropía. *UnivMon* [61] combina múltiples *Count Sketch* (CS) [27] en el plano de datos y búferes en el plano de control. Utiliza la cota superior $\sum m_i^2 > \sum m_i \log m_i$ para estimar la entropía, es decir la suma de los cuadrados de las frecuencias de paquetes m_i para cada flujo i. Sus resultados mostraron un error de estimación de aproximadamente el 2 % utilizando un *sketch* de 1 MiB. En [62], los autores propusieron *ElasticSketch* que combina una tabla *hash* que calcula las frecuencias de los flujos más frecuentes y un *sketch Count Min* (CM) [64] para los menos frecuentes. Ante colisiones se utiliza un esquema de votación que mueve un flujo de la tabla *hash* al *sketch*. *LightGuardian* [63] propuso una estructura de datos pequeña y de tamaño constante, llamada *sketchlet*. Un *sketchlet* es un fragmento del *sketch* presente en los dispositivos de red, que se transporta *in-band*, es decir transportan información en cada encabezado de paquete. En el *host* de destino, *LightGuardian* recolecta los *sketchlet*, reconstruye el *sketch* original y obtiene mediciones precisas de todos los flujos. Especificamente diseñaron el *sketch SuMax* que puede dividirse en *sketchlets* y estima la suma y los valores máximos de los flujos. *SuMax sketch* está basado en un CM pero con una política de actualización conservadora. Tanto *ElasticSketch* como *LightGuardian* estiman la entropía utilizando la distribución de flujos obtenida a partir de sus conteos estimados de paquetes. Logran errores menores al 1 % usando *sketches* de entre 1 y 3 MiB, pero han sido validados con conjuntos de datos de menos de un millón de flujos y menos de 6 millones de paquetes.

En [31], los autores propusieron un método para detectar los flujos top-K y aproximar la entropía empírica del tráfico de red utilizando solo las frecuencias estimadas de dichos elementos. El algoritmo utiliza un *sketch Count Min with Conservative Update* (CM-CU) [28] junto con una cola de prioridad para estimar y almacenar las frecuencias de las 10,240 direcciones IP de origen más frecuentes. En trazas de red reales [32, 33], el error relativo en la estimación de la entropía alcanzó hasta el 14 %, dependiendo de la traza. En [34], presentamos una versión mejorada de este método. Esta versión añade un *sketch* HLL para estimar el número de flujos distintos en el intervalo de observación, asumiendo que las frecuencias de los flujos no almacenados en la cola siguen una distribución uniforme. Las modificaciones realizadas redujeron el tamaño de la cola de prioridad a 8,192 elementos. Los resultados mostraron una reducción del promedio del error al 1.65 %, con un máximo del 3 %. Ambos algoritmos [31, 34] operan en conjuntos de datos de decenas de millones de flujos y cientos de millones de paquetes. Sin embargo, utilizan como identificadores de flujo solo la dirección IP de origen, con el uso de identificadores más complejos la cantidad de flujos aumenta drásticamente y la suposición de una distribución uniforme para las frecuencias de flujo no medidos aumenta significativamente el error de estimación.

He *et al.* [65] propusieron *HistSketch* para monitorear la distribución de los identificadores

de flujo y el cálculo de entropía. Para los elementos frecuentes registra sus distribuciones con contadores dedicados y para los menos frecuentes comparte contadores para reducir el uso de memoria. Además, propone dos mecanismos de optimización: el mecanismo de descarte de histogramas o *histogram shedding* y la decodificación basada en ecuaciones. En el primer caso, conserva los intervalos del histograma con alta frecuencia; es decir, los intervalos pesados en la parte dedicada, mientras expulsa los intervalos restantes hacia la parte compartida. El segundo mecanismo compensa el error causado por el uso compartido de contadores. Construye un sistema lineal para la parte compartida con los identificadores y los intervalos registrados. Además, reduce los costos de almacenamiento, transfiriendo los identificadores y los intervalos a un servidor centralizado. Por lo que requiere acceder a un servidor remoto para obtener resultados. En el conjunto de datos CAIDA 2018, *HistSketch* estimó la entropía de las frecuencias en intervalos de 149,549 elementos distintos, usando 2 MiB de memoria, con un error relativo del 5 %.

2.3.2. Mediciones de entropía en ventanas deslizantes

Assaf *et al.* [66] propusieron *Sliding Window Approximate Measurement Protocol* (SWAMP) para estimar la pertenencia a conjuntos, la cardinalidad y la entropía en trazas de red. SWAMP implementó una ventana deslizante utilizando un búfer cíclico y un puntero para indicar la posición de inserción actual. Cuando llega un nuevo elemento, el identificador de flujo más antiguo en la ventana es reemplazado. Las frecuencias de los flujos se mantienen en una tabla *hash* compacta llamada TinyTable [67], disminuyendo la frecuencia del identificador que sale y aumentando la del que entra. Un contador adicional registra el número de identificadores distintos en la ventana y se actualiza cada vez que la frecuencia se reduce a 0 o aumenta a 1. En trazas de CAIDA 2013 y 2016 [32], los autores utilizaron una ventana de 2^{16} elementos y lograron una estimación precisa de la entropía. Sin embargo, la complejidad espacial depende del número de elementos en la ventana deslizante. Además, el uso de TinyTable hace que su implementación en *switches* programables sea compleja [68].

En [69], los autores también ofrecieron una solución integrada para la estimación de *heavy hitters*, cardinalidad y entropía en ventanas deslizantes. Los *heavy hitters* son los elementos más frecuentes en un flujo de datos que superan un umbral definido por el usuario [38]. El *framework* propuesto basado en *sketches* especifica el intervalo de tiempo de interés como un parámetro de consulta. El modelo de consulta por intervalo generaliza las ventanas deslizantes para proporcionar visibilidad a cualquier intervalo dentro de la ventana [70]. Los autores extendieron el *sketch* *UnivMon* [61] a este modelo para estimar la entropía usando histogramas exponenciales [14] y

smooth histogram [71]. Las estimaciones de entropía se usaron para detectar ataques inyectados en trazas de CAIDA 2018 [32], usando una ventana de 1 millón de paquetes. Identificaron un posible ataque cuando la estimación de entropía actual no estaba dentro de cinco desviaciones estándar de la entropía pasada. Su solución necesita al menos 56 MiB de memoria para detectar con precisión un ataque DDoS con un error relativo medio inferior al 0,3 %.

Frameworks como *Exponential Count-Min* (ECM) *sketch* [72], *Sliding Sketch* [73], y *Half-Sliding Sketch* (HSS) [74] se utilizan para medir múltiples propiedades del tráfico de red. El *sketch* ECM estima frecuencias, cuantiles y detecta *heavy hitters* en ventanas deslizantes basadas en elementos y tiempo en un flujo de datos distribuido. Combina el *sketch* CM con histogramas exponenciales [14], reemplazando los contadores por histogramas. Estos histogramas dividen la ventana deslizante en subventanas de diferentes tamaños y responden consultas resumiendo los resultados de dichas subventanas. *Sliding Sketch* se utiliza para consultas de pertenencia, frecuencia y detección de *heavy hitters*. Puede adaptar cualquier *sketch* que siga el modelo de *k-hash* al modelo de ventana deslizante, como CM-CU y CS. Su estructura se divide en múltiples arreglos asincrónicos y utiliza una operación de escaneo para eliminar información desactualizada. Cada *bucket* contiene dos campos: datos antiguos y datos nuevos. Los resultados experimentales mostraron que el error relativo absoluto de *Sliding Sketch* es aproximadamente 40 veces menor que el de ECM, en la estimación de frecuencias, cuando se utiliza una memoria de 2 MiB. El *framework* HSS es una mejora de *Sliding Sketch*, utiliza solo un campo en cada *bucket* y una estrategia de subestimación para las consultas de frecuencia. Los autores mejoraron la precisión en la estimación de frecuencia entre 1.5 y 9 veces en comparación con *Sliding Sketch*, usando la misma cantidad de memoria. Además, es aproximadamente 1.5 más rápido.

Los algoritmos *Unbiased Cleaning Sketch*, *Sliding HyperLogLog* (SHLL) y *wSketch* han sido propuestos para la estimación de frecuencias, cardinalidad y detección de flujos top-K, respectivamente [16, 75, 76]. En [16], los autores lograron una medición no sesgada del tamaño de los flujos; donde el valor esperado de las frecuencias estimadas $\hat{m}_i$ es igual al valor de las reales m_i, $E(\hat{m}_i) = m_i$. El *sketch* propuesto basa su funcionamiento en el CS y puede eliminar elementos obsoletos de manera equilibrada en una ventana de elementos o tiempo. Su estructura está formada por un conjunto de matrices, donde cada fila actúa como un estimador no sesgado y utiliza un puntero de escaneo que recorre todas las filas de las matrices de forma circular y uniforme. En sus experimentos, los autores utilizaron trazas de CAIDA y datos sintéticos con distribución de Zipf [32, 77], empleando como identificador de flujo el par de direcciones IP de origen y destino. En el conjunto de datos Zipf, el error relativo promedio disminuye para todos los flujos a medida que aumenta el sesgo de la distribución. En trazas de CAIDA 2018, el error relativo promedio es

3.92 veces menor que el de *Sliding Sketch*. In [75], los autores propusieron *Sliding HyperLogLog* (SHLL) para responder consultas de cardinalidad en ventanas de tiempo. El algoritmo SHLL es basado en *HyperLogLog* [29] y su estructura de datos principal también es un arreglo de *buckets*, pero que mantienen, en cada uno, una lista de máximos posibles futuros. Cuando se recibe un paquete, solo se actualiza la lista asociada y los paquetes que salen de la ventana son eliminados. En trazas de tráfico sintético y real, SHLL logró un error estándar similar al de HLL, pero necesita mayor almacenamiento en memoria. La memoria requerida depende del número de flujos presentes en la ventana de tiempo. Recientemente, se propuso *wSketch* para la detección en tiempo real de los flujos top-K en ventanas de tiempo [76]. Su estructura incluye un *sketch* para almacenar flujos frecuentes, un arreglo para flujos menos frecuentes y un modelo de cola circular que reutiliza el espacio en memoria. El algoritmo emplea un mecanismo de ponderación basado en probabilidades que asigna diferentes pesos a flujos grandes y pequeños, favoreciendo la retención de los flujos más grandes. Los resultados experimentales en un conjunto de datos público [78] demostraron una precisión superior al 96 % utilizando solo 20 KiB de memoria. Sin embargo, debido al uso de ventanas deslizantes y cálculos relativamente complejos, *wSketch* presenta limitaciones para su implementación en el planos de datos de *switches* programables.

2.4. Aceleración hardware

Los aceleradores hardware proporcionan la capacidad computacional para procesar grandes volúmenes de datos con alto rendimiento y baja latencia, pero su desempeño está limitado por la cantidad de memoria interna disponible en el dispositivo. La memoria interna ofrece un mayor ancho de banda agregado y paralelismo que accesos a memoria externa. Por lo tanto, los algoritmos que maximizan el uso de la memoria interna pueden alcanzar un rendimiento superior en estas plataformas [22]. En este contexto, se han empleado varios algoritmos de *streaming* basados en *sketches* para mediciones de tráfico de red [23, 24, 25]. Estos algoritmos son implementables en hardware debido a su uso sublineal de memoria, su alto grado de paralelismo intrínseco y a que ofrecen garantías teóricas sobre el error de estimación.

2.4.1. Aceleración hardware usando P4

Los dispositivos de red modernos incluyen tarjetas inteligentes de interfaz de red (*Network Interface Cards*, NICs), y *switches* y *routers* programables, que poseen un procesador de pro-

pósito específico capaz de extraer información de los paquetes entrantes y procesarla según la aplicación [17]. Dicho procesador suele ser programado mediante un lenguaje específico de dominio (*Domain-Specific Language*, DSL) y puede implementarse en FPGA o como un circuito integrado personalizado. Pueden procesar miles de millones de paquetes por segundo, con latencias del orden de los microsegundos [17]. Los DSL imponen limitaciones adicionales, tanto en el modelo de programación como en las operaciones disponibles. Por ejemplo, P4 que actualmente es el lenguaje más utilizado en el plano de datos, no admite iteraciones, aritmética en punto flotante y operaciones como la división, los logaritmos y exponenciales [18].

Varias investigaciones estiman la entropía de flujos de red en el plano de datos de los *switches* programables. En [79], los autores propusieron un mecanismo basado en la entropía de Shannon para detectar y mitigar ataques DDoS utilizando P4. Su enfoque estimó la distribución de frecuencias de las direcciones IP de origen y destino de los paquetes mediante el muestreo de *Alon-Matias-Szegedy* [57] y un *Count Sketch* [27]. Dado que P4 no cuenta con funciones logarítmicas nativas, utilizaron tablas de búsqueda por coincidencia de prefijo más largo (*Longest Prefix Match*, LPM) y almacenaron los resultados en memoria TCAM. Su implementación alcanzó una precisión de detección del 98.2 % con una latencia de 250 ms. Este enfoque sobrecarga los limitados recursos de TCAM del *switch* y restringe el número de paquetes en la ventana de observación a una potencia de dos. Los autores extendieron su trabajo en [80] para mitigar ataques, utilizando un criterio basado en umbrales para descartar, limitar o redirigir paquetes. En [81], los autores emplearon el algoritmo propuesto en [79] para diseñar un mecanismo colaborativo para la mitigación de ataques DDoS. Este mecanismo usa la entropía de las direcciones IP de los paquetes para identificar direcciones sospechosas y alertar a otros *switches*. Los dispositivos filtran los paquetes y propagan la alerta, confinando eventualmente los flujos de ataque cerca de su origen. El algoritmo alcanzó hasta un 94 % de precisión y una latencia significativamente menor de 0.1 ms.

Lai *et al.* [53] implementaron en P4 el algoritmo de estimación de entropía propuesto por Clifford y Cosma [12]. Los autores estimaron la entropía de las direcciones IPv4 de origen utilizando un contador de paquetes y un *sketch* de estimación de frecuencias con proyecciones aleatorias de 2,048 KiB; los resultados de estas estructuras se procesaron para calcular el valor estimado de entropía en el plano de control. Evaluaron su implementación en dos trazas: MawiLab 2007 con aproximadamente 18 millones de paquetes y MawiLab 2015 con 147 millones [33]. Los resultados mostraron que el error absoluto medio fue de 8.5 % y 11.85 %, respectivamente. Su implementación se ejecutó en un *switch* Barefoot Tofino a una velocidad de 100 Gbps.

Ding *et al.*[35] propusieron un algoritmo para estimar la entropía empírica del tráfico de red

y detectar ataques DDoS en un entorno emulado de *switches* P4. Para la estimación de entropía de los flujos identificados por la misma dirección IP de destino utilizaron un CS y las primitivas P4Log y P4Exp [18]. En trazas de CAIDA, divididas en diez intervalos de tiempo, cada uno con un número fijo de 2^{21} paquetes, sus resultados mostraron un error relativo medio del 3 % en la estimación de entropía. La estimación se realizó completamente en el plano de datos sin requerir memoria TCAM y el CS requirió 40 KiB.

En [56], los autores propusieron un método de interpolación tabular que estima la entropía de los flujos de red utilizando el *framework* de proyecciones aleatorias estables [60]. El método utiliza una técnica de muestreo para construir una función de distribución empírica en una tabla de búsqueda de solo lectura. Los autores redujeron el tamaño de la tabla mediante una heurística de interpolación lineal por tramos. El algoritmo alcanzó un error relativo de estimación de hasta 3.6 % en trazas de MawiLab con aproximadamente 1.7 millones de flujos y 196 millones de paquetes. La implementación en un *switch* Tofino2 se ejecutó a 400 Gbps.

En [82] presentamos un algoritmo de estimación de entropía basado en [34], que utiliza un *sketch* CM-CU y un arreglo de colas de prioridad para rastrear las fecuencias de los top-K flujos, así como un *sketch* HLL++ [83] para estimar el número total de flujos. El algoritmo combinó operaciones de alta velocidad en el plano de datos con procesamiento no crítico en el plano de control, en este último se ejecuta únicamente el paso final de la estimación de entropía. La aproximación logaritmica utilizada se basó en el algoritmo P4Log [18], pero con modificaciones para la parte freccionaria del número, utilizando una tabla de búsqueda en lugar de un árbol binario. Implementado en el modelo de comportamiento P4 BMv2, con 12 trazas de los repositorios públicos CAIDA [32] y MawiLab [33], en diferentes intervalos de observación el algoritmo logró un error de estimación inferior al 3.26 % y en promedio de 1.72 %.

Zhang *et al.* [84] propusieron un mecanismo para detectar ataques DDoS basado en la entropía, implementado en *switches* programables utilizando el modelo de ventanas deslizantes. Este enfoque combina la información de entropía con un árbol de decisión. Para la detección inicial, calcularon la entropía de las direcciones IP y utilizan un umbral adaptativo en cada ventana. Si se detecta un paquete de ataque, este se envía al módulo de redetección basado en aprendizaje automático. Utilizaron el CM *sketch* [28] para aproximar las frecuencias de las direcciones IP de los paquetes y una tabla de búsqueda de LPM para calcular la función logarítmica. En trazas de CAIDA 2016 y CAIDA DDoS Attack 2007, el mecanismo logró la detección con más del 96 % de verdaderos positivos y menos del 3 % de falsos positivos bajo ataques de tipo ICMP flood, ACK flood, SYN flood y UDP flood.

2.4.2. Aceleración hardware basada en FPGA

Aceleradores hardware implementados en Circuitos Integrados de Aplicación Específica (*Application Specific Integrated Circuits*, ASIC) o en FPGA son usados en la estimación de entropía de tráfico de red [53, 85, 20, 86, 31, 34]. En especial, los FPGA son ampliamente utilizados en diferentes mediciones de tráfico por tres razones principales. En primer lugar, ofrecen un alto grado de flexibilidad, lo que permite adaptar la configuración del hardware a características específicas de los flujo de datos o a los requisitos del usuario. Por ejemplo, algunas configuraciones pueden optimizarse para obtener mayor precisión, mientras que otras priorizan un mayor *throughput*. En segundo lugar, los algoritmos de medición se basan principalmente en el uso de *sketches*. Los FPGAs modernos disponen de memoria SRAM, integrada en el chip para almacenar estas estructuras de datos de manera eficiente, sin necesidad de acceder a la DRAM externa, que es más lenta. En tercer lugar, ofrecen un amplio soporte y capacidades de entrada/salida de alto ancho de banda y gran flexibilidad, lo que facilita su integración en aplicaciones tanto de borde como en la nube, destinadas al análisis de grandes flujos de datos.

En [85], los autores ampliaron el trabajo de Lall *et al.* [11] proponiendo un esquema basado en CS para estimar la entropía, en lugar de utilizar un procedimiento de conteo exacto. La entropía es calculada en el *host* basada en las frecuencias estimadas por el CS de dimensiones $3 \times 32,768$. Los resultados experimentales en trazas de CAIDA 2012 [32], en períodos de observación de más de 30 millones de paquetes y 448,809 flujos identificados por la dirección IP de origen muestran un error relativo inferior al 1.5 %. El esquema se implementó en un FPGA Xilinx Virtex-5 de la plataforma NetFPGA-10G y requirió tres ciclos de reloj para aplicar la función *hash* al identificador de flujo entrante de 32 bits y actualizar el CS. Soportó velocidades de 30 Gbps con una frecuencia de reloj de 160 MHz.

Wellem *et al.* [20] propusieron *FlexSketchMon*, un *framework* que soporta algoritmos basados en *sketches* para diferentes mediciones de tráfico, incluida la entropía. En el plano de datos se estimaron y almacenaron las frecuencias de los flujos, mientras que en el plano de control se calculó la entropía. El hardware del plano de datos incluyó una tabla *hash* que almacena contadores de paquetes y bytes de los flujos durante el período de observación, una tabla que almacena los identificadores y un filtro Bloom. Estas estructuras de datos son demasiado grandes para almacenarse en memoria interna; por lo que se utilizaron memorias externas QDRII+SRAM y DDR3 DRAM. En trazas de CAIDA 2012 con intervalos de observación de 15 segundos el error de estimación de entropía es inferior al 0.4 %. La implementación en la plataforma NetFPGA-SUME alcanzó velocidades de 96 Gbps con paquetes Ethernet de tamaño mínimo.

Lai *et al.* [86] presentaron un sistema de detección en tiempo real de ataques DDoS usando estimaciones de entropía y una red neuronal recurrente con memoria a largo y corto plazo (*Long Short-Term Memory Recurrent Neural Network*, LSTM-RNN) en la plataforma NetFPGA-10G SUME. Para la estimación de entropía se utilizó el método de Clifford y Cosma [12], discutido en la sección anterior. Los cálculos costosos como logaritmos de los valores aleatorios con distribuciones α-estables se precalcularon y almacenaron en tablas de consulta en el FPGA. En el plano de control la LSTM-RNN analizó la evolución temporal de las mediciones de entropía y detectó ataques a velocidades de 40 Gbps. Los experimentos en trazas de CAIDA 2007 y MawiLab 2007, 2015 y 2019 mostraron una precisión en la detección de ataques mayor al 91 %.

En [31], se presentó un acelerador hardware que estima la entropía de los elementos más frecuentes en el flujo de red utilizando un *sketch CM-CU* de $16,384 \times 4$ y una cola de prioridad de 10,240 elementos. Implementado en una FPGA Xilinx Zynq UltraScale + ZCU102 lograron un rendimiento de un paquete por ciclo a 354 MHz, lo que le permite operar a velocidades de hasta 181 Gbps con paquetes Ethernet de tamaño mínimo. Posteriormente, los mismos autores, presentaron en [34] un método más preciso para estimar la entropía de un flujo de datos. El método utiliza un *sketch* HLL para estimar la cardinalidad del flujo y un *sketch CM-CU* de $16,384 \times 8$ contadores para estimar las frecuencias de los K elementos más frecuentes en línea y un arreglo de colas de prioridad para almacenar las frecuencias top-K. Luego estiman la contribución de entropía de los flujos restantes asumiendo una distribución uniforme simple. Este método se implementó en una FPGA Xilinx UltraScale + ZCU102; usando solo memoria la memoria interna, con menos del 50 % de uso de recursos. Probado en trazas de CAIDA, MawiLab y Mendeley, de hasta 120 millones de paquetes y más de 5 millones de flujos, el acelerador estima la entropía empírica con menos del 1,5 % de error relativo medio, una frecuencia de reloj de 400 MHz y admite un rendimiento mínimo de 204 Gbps.

Otros algoritmos de medición de tráfico también han sido propuestos para su implementación en FPGAs [87, 19, 88]. En [87], los autores modificaron la técnica de detección de *heavy changes* basada en *K-ary Sketch* [89]. Los *heavy changes* son aquellos elementos que experimentan cambios abruptos en su frecuencia entre dos ventanas de tiempo consecutivas [42]. Propusieron una arquitectura completamente paralelizada basada en *sketches* sobre FPGA para acelerar el algoritmo modificado. La arquitectura predice la actividad de las entidades de red según su historial, y luego reporta aquellas cuya diferencia entre la actividad observada y la actividad predicha excede un umbral establecido. La implementación en un FPGA Xilinx Virtex-7 xc7vx690tffg1930-3 alcanzó velocidades entre 96 y 103 Gbps, utilizando diversas configuraciones para la detección de *heavy changes* en línea. Posteriormente, en [19] los mismos autores

propusieron una arquitectura para acelerar dos algoritmos en línea: CM *sketch* y *K-ary Sketch*, orientados a la detección de *heavy hitters* y *heavy changes*. La arquitectura fue implementada en un FPGA Xilinx Virtex UltraScale xcvu440, y operó sobre enlaces de 150 Gbps para la detección de *heavy hitters* y de 100 Gbps para *heavy changes*. La frecuencia máxima de reloj alcanzada fue de 477 MHz. Los autores demostraron que el acceso paralelizado a tablas *hash*, mejora significativamente el rendimiento de la arquitectura.

En [88], se propuso *Jigsaw-Sketch* para detectar flujos top-K. *Jigsaw-Sketch* utiliza un arreglo de *buckets* o celdas para identificar los flujos de mayor tamaño y una estrategia de reemplazo probabilístico para conservar los flujos grandes y reemplazar los pequeños por otros nuevos. Seguido almacena los identificadores de los flujos top-K candidatos en una lista auxiliar. Los autores diseñaron un mapeo de posición uno a uno, que permite que cada flujo capturado localice directamente su posición de almacenamiento según la posición en la que se captura. Implementaron el algoritmo en una NetFPGA-1G-CML usando 200 KiB de memoria y alcanzaron una frecuencia de reloj de 115,55 MHz. En trazas de CAIDA 2016 y 2019, a medida que K aumenta de 1000 a 5000, el error relativo promedio aumenta de 0.003 a 0.014 y de 0.001 a 0.008, respectivamente. En todos los casos, su precisión es superior a 0.96.

Mediciones del tamaño de los flujos o estimaciones de frcuencias usando el modelo de ventana deslizantes también han sido implementadas en FPGA [90, 91]. En [90], los autores presentaron cuatro estrategias de implementación para el ECM *sketch*, cada una con distintos compromisos entre costo y rendimiento. Las arquitecturas propuestas fueron evaluadas en una FPGA Virtex-6, alcanzando frecuencias de operación entre 170 MHz y 260 MHz y tasas de procesamiento de entre 165 y 260 millones de tuplas por segundo. En [91], los autores implementaron *Sliding Sketch* en una FPGA Xilinx Alveo U280 para acelerar el cálculo de la dirección del *bucket* mapeado y la corrección de los resultados basada en la distancia entre el *bucket* y el puntero de escaneo. Con las modificaciones realizadas aceleraron la actualización y la velocidad de consulta del *sketch* deslizante a un máximo de 45,8 veces sin afectar la precisión. Los autores evaluaron el efecto de la estrategia de corrección utilizando una ventana de 1 millón de direcciones IP fuentes y una memoria de 5 MiB. Con la corrección, el error relativo promedio de *Sliding Sketch* CM-CU se redujo en un 12 %.

2.5. Discusión

La entropía empírica es una medición de red utilizada en aplicaciones de monitoreo de tráfico como la detección de anomalías [11, 49]. Es utilizada comunmente para detectar ataques de DDoS [81, 86]. Para su estimación es necesario considerar varios identificadores de flujo, el volumen, velocidad y distribuciones de tráfico. Dependiendo de las necesidades de la aplicación con respecto a la antigüedad de los datos y el período de tiempo consultado, pueden utilizarse modelos basados, principalmente, en ventanas fijas o ventanas deslizantes.

En la estimación de entropía en grandes volúmenes de datos son utilizados los algoritmos basados en *sketches*, porque comprimen y registran varias estadísticas de los datos. La precisión y el rendimiento de latencia se correlacionan con sus requerimientos de memoria, y por tanto con sus dimensiones. En redes, las frecuencias de los flujos suelen seguir una distribución sesgada, con un rango de valores que abarca varias órdenes de magnitud, por lo que suelen requerirse *sketches* de gran tamaño. Ante estas limitaciones, para la estimación de entropía varias investigaciones utilizan *sketches* que diferencian entre elementos frecuentes y menos frecuentes. Otros utilizan *sketches* de frecuencias y una cola de prioridad para estimar las frecuencias top-K y supuestos sobre la distribución de tráfico para estimar el resto de las frecuencias. Las investigaciones analizadas no logran un equilibrio entre errores de precisión menores al 3 % y un uso eficiente del espacio en memoria, considerando varios identificadores de tráfico en escenarios reales que incluyan decenas de millones de flujos y cientos de millones de paquetes. En el modelo de ventana deslizante, los *sketches* presentan mayores requerimientos de memoria para lograr eliminaciones de grano fino de datos antiguos, porque generalmente necesitan almacenar los identificadores de flujos. Por lo tanto, estimaciones precisas de entropía usando *sketches* requieren un mayor consumo de espacio en memoria. Los algoritmos analizados son escasos, estos estiman la entropía de los flujos en ventanas deslizantes basadas en elementos, no consideran ventanas de tiempo; y no logran un equilibrio entre precisión y uso de memoria.

Tecnologías como *switches* programables y FPGAs permiten capacidades de procesamiento de alta velocidad dentro de la red. Los *switches* programables experimenta el uso del lenguaje P4. Una alternativa para mejorar el rendimiento general de las redes, dada su reconfigurabilidad, independencia del protocolo e independencia de la plataforma. Su uso se ha extendido a varios dispositivos como FPGA, que tienen la ventaja de ser reprogramables, lo que añade flexibilidad al flujo de diseño. Los aceleradores que usan solo recursos internos del chip pueden lograr un mayor rendimiento en comparación con las soluciones que requieren un acceso frecuente a memoria externa; por tanto, son ampliamente abordados en la literatura.

Capítulo 3. Estructuras de datos

3.1. Introducción

En este capítulo se presenta el marco teórico de la investigación, asociado a las estructuras de datos utilizadas en los algoritmos y aceleradores hardware diseñados. Se describen las estructuras de datos basadas en *sketches* usadas para estimar la frecuencia y cardinalidad de los flujos de red. Además, se describe la estructura utilizada para aproximar el comportamiento de una cola de prioridad, que almacena las frecuencias de los flujos top-K. Los *sketches* y la cola de prioridad aproximada emplean la función *Murmurhash3* para distribuir uniformemente los datos en la estructura, por lo que también es analizada.

3.2. Funciones *hash*

Las funciones *hash* mapean datos a un espacio de menor dimensión, bajo las siguientes condiciones. Primero, cada elemento se asigna de forma aleatoria o estratégicamente a una posición dentro del espacio de *hash*. Segundo, dos elementos con los mismos valores generarán el mismo valor *hash* y se asignarán a la misma posición. El término estratégicamente hace referencia al uso de funciones *hash* diseñadas con propiedades específicas que favorecen el comportamiento del algoritmo en el que se integran. Tales propiedades incluyen la uniformidad en la distribución de los datos o la baja probabilidad de colisiones. Las colisiones se producen cuando elementos distintos pueden generar el mismo valor *hash*. Estas son inevitables si hay más registros de datos que posiciones disponibles en el espacio de *hash*. En consecuencia, las funciones *hash* enfrentan principalmente dos desafíos: el diseño de funciones eficaces que minimicen las colisiones o aumenten la precisión sin pérdidas de eficiencia, y el manejo de colisiones cuando ocurren [92].

Las técnicas de *hashing* han evolucionado desde enfoques simples de aleatorización hasta métodos adaptativos avanzados que consideran la localidad, la estructura y la seguridad de los datos. En aplicaciones con datos de entrada de gran tamaño que requieren búsquedas o procesamiento rápido es ampliamente utilizada la función *MurmurHash3* [93]. Es una función *hash* no criptográfica, no está diseñada para seguridad, sino para eficiencia. Los métodos en esta categoría comparten alta velocidad de cómputo, baja probabilidad de colisiones, alta probabilidad de detección de errores y facilidad para detectar colisiones. La función *MurmurHash3* de 32 bits,

debido a su estructura simple que utiliza una secuencia de multiplicaciones, sumas, operaciones lógicas y desplazamientos de bits, ha sido previamente paralelizadas en el diseño de aceleradores hardware para la estimación de entropía usando *sketches* [34].

3.3. Estructuras probabilísticas: *sketches*

Los *sketches* son estructuras de datos probabilística que resumen un conjunto de datos. Se componen, generalmente, de varias tablas *hash* o arreglos, que aleatorizan su contenido y lo representan en un tamaño fijo, obteniendo una cantidad determinada de bytes para una entrada de datos de cualquier tamaño. Los *sketches* realizan diferentes operaciones como: i) actualización para comprimir y registrar información, ii) consulta para proporcionar resultados estadísticos, iii) combinación o unión para fusionar varios *sketches* en uno solo y iv) reversibilidad para obtener de forma inversa determinada identificador de flujo [94]. La forma de realizar estas operaciones difiere de un *sketch* a otro y no todos incluyen la unión y/o reversibilidad.

3.3.1. Estimación de cardinalidad

El *sketch* HyperLogLog es utilizado para estimar la cardinalidad de los flujos en el tráfico de red [29] y ha sido implementado en hardware dedicado [34]. Tiene una complejidad espacial sublineal y proporciona un error de estimación bajo tanto en teoría como en la práctica [29, 83]. En la Figura 3.1 se muestra su estructura, un arreglo A de $m = 2^p$ celdas, las cuales son enteros inicializados en cero. El parámetro p define el tamaño del arreglo y determina la precisión de la estimación de cardinalidad. El *sketch* soporta las operaciones de inserción, consulta y combinación.

La inserción actualiza el contenido del *sketch* para cada nuevo elemento e en el *stream*. Primero, se calcula una función hash h sobre el valor de e, en nuestra implementación, e corresponde al identificador de flujo de cada paquete. Los p bits más significativos (v_1) del valor *hash* $h(e)$ se utilizan para seleccionar una celda en el arreglo. Luego, se calcula la posición del bit más significativo en uno (o, equivalentemente, el número de ceros iniciales más uno, es decir, $ldz(v_2) + 1$) en los $b = |h(e)| - p$ bits restantes del valor *hash*. Si este valor es mayor que el contenido actual a de la celda correspondiente, se reemplaza el valor almacenado en el *sketch*, como se muestra en la Figura 3.1. El número de bits requerido para representar cada celda en A es $\lceil \log_2(b+1) \rceil$.

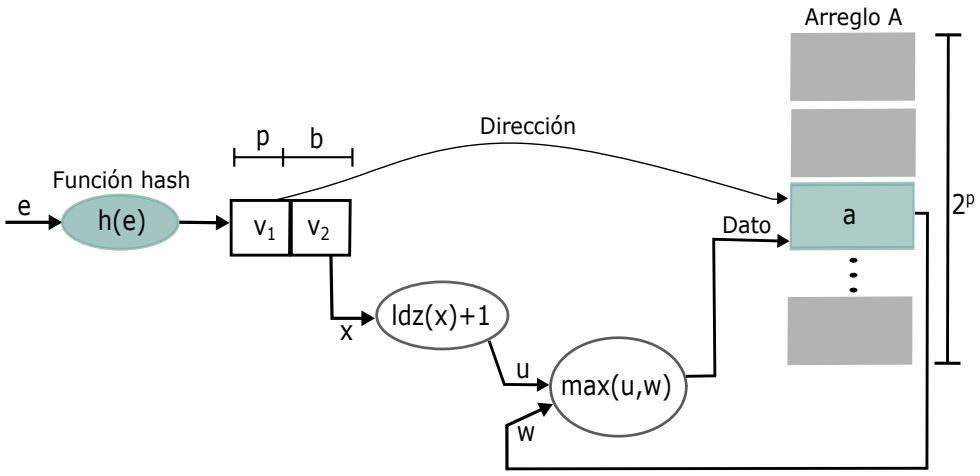


Figura 3.1: Estructura del *sketch* de cardinalidad HyperLogLog.

HLL calcula la media armónica Z de los elementos en A:

$$Z = \left(\sum_{i=0}^{|A|-1} 2^{-A[i]} \right)^{-1}, \tag{3.1}$$

y la la cardinalidad estimada $\hat{N}$ como:

$$\hat{N} = \beta_A |A|^2 Z, \tag{3.2}$$

donde β_A es un factor de corrección que depende del número de celdas en el arreglo [29], definido como:

$$\beta_A = \left(A \int_0^{\infty} (log_2(\frac{2+a}{1+a}))^m \, du \right)^{-1}, \tag{3.3}$$

donde m es la cantidad de celdas y a son los elementos del conjunto. Para valores de m mayores o iguales a 2^7, β_A puede ser aproximado a $\frac{0{,}7216m}{m+1{,}079}$. Con esto, HyperLogLog reduce el error estándar a $\delta \approx \frac{1{,}04}{\sqrt{m}}$. En flujos con baja cardinalidad en comparación con $|A|$, la mayoría de las celdas en A son cero y la Ecuación (3.2) produce un error de estimación elevado. Heule *et al.* [83] demostraron que, cuando $\hat{N} \leq 2{,}5|A|$, la cardinalidad se puede estimar mejor como:

$$\hat{N} = |A| \log \left(\frac{|A|}{n_z} \right), \tag{3.4}$$

donde n_z es el número de celdas con valor cero. Heule *et al.* proposieron una versión mejorada de HLL, llamada *HperLogLog++* [83] para el caso de *sketches* de baja ocupación. Sin embargo, en casos de ocupación normal HyperLogLog++ y HLL presentan una estimación idéntica.

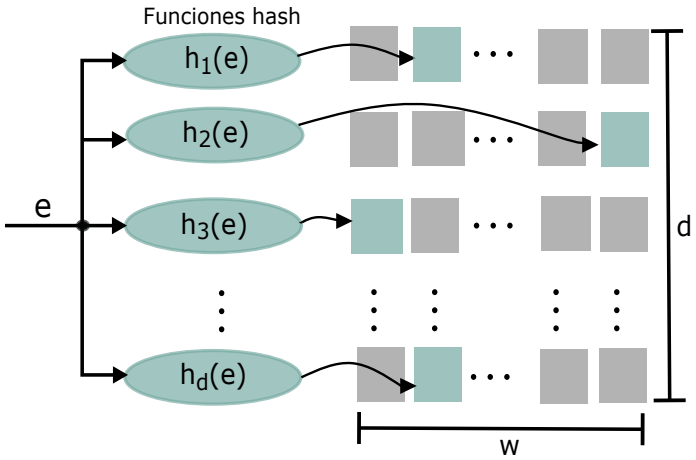


Figura 3.2: Estructura genérica de *sketches* de cuentas o de estimación de frecuencias.

3.3.2. Estimación de frecuencias

En la Figura 3.2 se muestra la estructura genérica de un *sketch* de cuentas utilizado para estimar la frecuencia de los paquetes. La estructura de datos principal es un arreglo de $d \times w$ celdas, donde cada una es un entero no negativo que almacena una cuenta, y se cumple que $(d \times w) \ll N$, siendo N el número de elementos o flujos distintos. Proporciona dos operaciones básicas: 1) inserción, que actualiza el contenido del *sketch* por cada nuevo elemento en el flujo, por ejemplo un paquete de red; y 2) consulta, que entrega una estimación del número de veces que se ha visto ese elemento en particular.

Cada tipo específico de *sketch* define sus propias operaciones de inserción y consulta, pero en general, las celdas se inicializan en cero y, en cada actualización o consulta, el *sketch* utiliza un conjunto de d funciones hash h_i, con $i \in [1, d]$, para asignar una celda de cada fila al elemento entrante, como se ilustra en la Figura 3.2.

En este trabajo, evaluamos dos tipos de *sketches* de conteo:

1. CM-CU [28, 95]: La operación de actualización toma los d contadores seleccionados por las funciones *hash* e incrementa en uno aquellos con el valor mínimo, es decir, los que han experimentado la menor cantidad de colisiones. La consulta estima el conteo del elemento como el mínimo de los d contadores seleccionados.

2. CS [27]: En la operación de actualización, el *sketch* aplica un segundo conjunto de funciones *hash* g_i sobre el elemento del flujo para decidir si se incrementa o decrementa en uno el contador seleccionado. La consulta estima el conteo del elemento como la mediana de los d

contadores. CS puede generar una sub o sobre estimación de la frecuencia de un elemento debido al algoritmo de inserción que utiliza.

Ambos *sketches* no son reversibles pero soportan la operación de unión o combinación y han sido implementados en hardware dedicado.

3.4. Arreglo de colas de prioridad para detección de top-K

Mantener una cola de prioridad exacta con K elementos en software es computacionalmente costoso, ya que cada operación de inserción o actualización requiere al menos $\log_2 K$ accesos a memoria. Este sobrecosto la hace impráctica para aceleradores hardware que deben procesar un paquete por ciclo de reloj. Para abordar esta limitación, en [34], se aproximó el comportamiento de una cola de prioridad grande organizando los K_{pq} elementos en un arreglo de R colas de prioridad más pequeñas, cada una con S elementos ($R = K_{pq}/S$), estructura que se denominó arreglo de colas de prioridad (*Priority Queue Array*, PQA). Cada elemento en el PQA es una tupla que contiene una etiqueta de flujo y su frecuencia, donde esta última determina la prioridad del elemento dentro de su cola de S elementos. Todas las frecuencias se inicializan en cero.

En la Figura 3.3 se muestra la estructura del PQA. Para cada nuevo elemento de datos se actualiza su contenido, utilizando el identificador de flujo e del paquete y su frecuencia estimada m_e, obtenida del *sketch* de cuentas. Primero, el PQA calcula una función *hash* h sobre el identificador de flujo. Los $\log_2 R$ bits menos significativos de $h(e)$ se utilizan para seleccionar una cola ordenada dentro del arreglo. Luego, se inserta la tupla, que consiste en la frecuencia del flujo m_e y una etiqueta con los bits restantes de $h(e)$ en la cola seleccionada.

Si m_e es menor que todos los elementos en la cola, la tupla se descarta. De lo contrario, se inserta en su lugar correspondiente y se descarta la tupla con la frecuencia más pequeña de la cola. Dado que $S \ll K_{pq}$, se puede realizar esta operación en un solo paso, o como una actualización de un solo ciclo en un acelerador hardware. Además, esta estructura puede implementarse eficientemente en aceleradores basados en FPGA utilizando módulos de memoria BRAM (Block RAM) para cada una de las S colas, los cuales pueden accederse en paralelo para insertar un nuevo elemento en la cola correspondiente.

Dado que la función *hash* distribuye los elementos entrantes de manera uniforme a lo largo del arreglo, el contenido del PQA aproxima al de una cola de prioridad real. Sin embargo, los

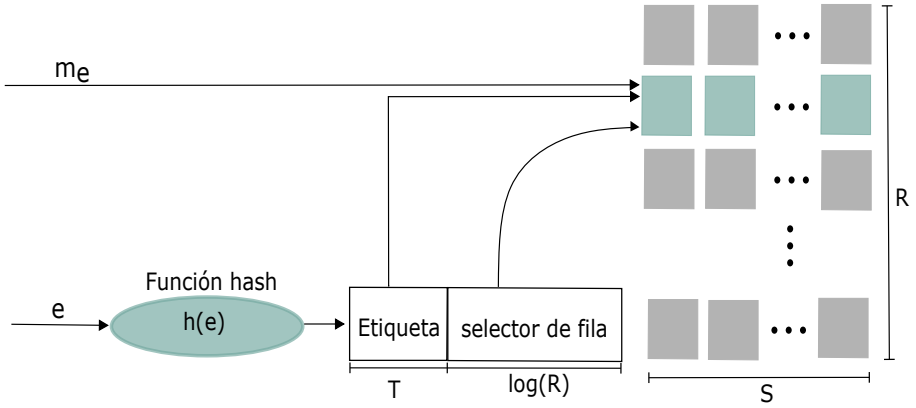


Figura 3.3: Estructura del arreglo de colas de prioridad.

elementos solo están ordenados localmente dentro de cada cola de S elementos. Por lo tanto, el contenido del PQA no está ordenado globalmente. También es posible que el número K de elementos válidos en el PQA sea menor que K_{pq}.

3.5. Discusión

Los *sketches* permiten representar grandes volúmenes de datos de manera compacta, con un uso de memoria sublineal, lo cual resulta esencial en entornos donde el tráfico de red es masivo y continuo. Estas estructuras presentan un alto grado de paralelismo, lo que favorece su implementación en aceleradores hardware. Además, ofrecen un balance entre precisión y complejidad: por un lado, reducen significativamente el consumo de recursos en comparación con los métodos exactos; por otro, introducen un error acotado que se mantiene dentro de márgenes aceptables para aplicaciones de detección de anomalías y análisis de tráfico. En conjunto, la combinación de *sketches* y el arreglo de colas de prioridad analizado constituye una base teórica y práctica sólida para el diseño del primer algoritmo propuesto, así como para su implementación en un acelerador hardware en FPGA, aspecto que se desarrolla en el capítulo siguiente.

Capítulo 4. Entropía en tiempo discreto

4.1. Introducción

En el presente capítulo se define la entropía de Shannon y la entropía empírica normalizada. Se presenta un método y algoritmo para estimar la entropía empírica de flujos de red en un intervalo de tiempo discreto, usando la ley de potencias. El algoritmo propuesto utiliza estructuras de datos como *sketches* para estimar la cardinalidad y frcuencia de los flujos y un arreglo de colas de prioridad para detectar y almacenar los flujos top-K. Además, se propone y evalua el rendimiento de una arquitectura en FPGA para su aceleración hardware. El algoritmo y acelerador se evalúan en un conjunto de trazas reales disponibles en repositorios públicos.

4.2. Entropía empírica de Shannon

La definición matemática de la entropía de Shannon de una variable aleatoria discreta X, se define como:

$$H_s(X) = -\sum_{i=1}^{c} p_i \ \log p_i, \tag{4.1}$$

donde p_i es la probabilidad de ocurrencia del valor i y c es el número de valores distintos posibles que puede tomar la variable X [1]. Cuando se estima la entropía a partir de un conjunto de datos obtenido mediante un número finito de muestras de un proceso aleatorio, como los flujos de red registrados durante un intervalo de observación dado, la entropía empírica de Shannon puede calcularse como:

$$H(X) = -\sum_{i=1}^{c} \hat{p_i} \log \hat{p_i}, \tag{4.2}$$

donde $\hat{p_i} = m_i/M$ es la estimación de la probabilidad p_i del valor i-ésimo, m_i representa el número de ocurrencias (o frecuencia) del valor i-ésimo, y es el número total de muestras dentro del intervalo de observación.

En aplicaciones de monitoreo de redes, $m_i = 0$ para la mayoría de los valores de la variable durante un intervalo de observación. Por lo tanto, la Ecuación (4.2) puede expresarse como:

$$H(X) = -\sum_{i=1}^{N} \hat{p}_i \ \log \hat{p}_i, \tag{4.3}$$

donde N es el número total de flujos distintos en el intervalo de observación [53]. Dado que el valor de H depende de N [49], el cual puede variar incluso entre distintos intervalos de observación de un mismo proceso, es común normalizar la entropía respecto a su valor máximo para poder comparar conjuntos de datos de diferentes tamaños. La entropía normalizada se define como:

$$H_{norm} = \frac{H}{\log N} \tag{4.4}$$

y toma valores en el rango entre 0 y 1.

Calcular la entropía empírica según las Ecuaciones (4.3) y (4.4) requiere determinar los valores de N, M y m_i para cada elemento distinto en el conjunto de datos. El valor de M puede obtenerse contando el número total de elementos en el conjunto. Sin embargo, calcular los valores exactos de N y de cada m_i es más complejo, ya que ambos requieren mantener estructuras de datos (por ejemplo, un *heap* de indicadores o contadores) cuyo tamaño es proporcional al número total de elementos distintos. En aplicaciones de monitoreo de red, los elementos suelen ser flujos que se identifican mediante los cinco campos del encabezado de cada paquete, que puede generar conjuntos de datos con cardinalidades del orden de decenas de millones de flujos distintos. Para calcular la entropía en tiempo real, estas estructuras de datos deben actualizarse cientos de millones de veces por segundo en enlaces de red que soportan velocidades de línea superiores a 100 Gbps. Para alcanzar este rendimiento, los *switches* programables y los aceleradores hardware deben almacenar los datos en la memoria interna (*on-chip*), la cual es un recurso escaso en estos dispositivos. Sin embargo, en muchas aplicaciones es suficiente estimar la entropía, lo cual puede realizarse usando estructuras de memoria sublineales. El desempeño de la aplicación dependerá en gran medida del error de estimación que pueda alcanzar el algoritmo [53, 35, 56, 20].

4.3. Método

En la Figura 4.1, la curva roja muestra un histograma normalizado representativo de las probabilidades de los flujos, obtenido a partir de datos reales del repositorio MawiLab [33]. Este intervalo de observación contiene más de 26 millones de flujos y 76 millones de paquetes. El método propuesto se basa en dos propiedades observadas en los conjuntos de datos, presentes en múltiples trazas provenientes de distintos repositorios [33, 32, 96]. Primero, en el tráfico de

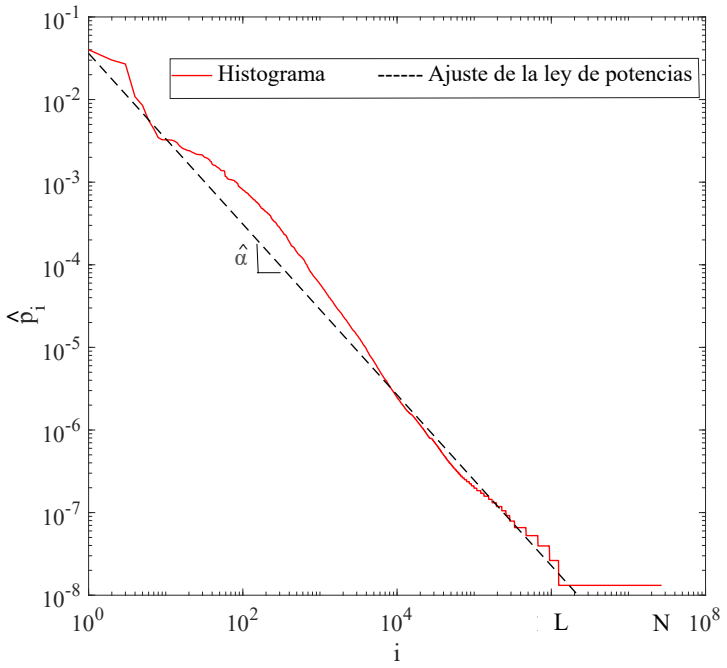


Figura 4.1: Histograma normalizado de probabilidades de los flujos para la traza Mawi-20181400 en escala log-log. La curva roja representa las frecuencias normalizadas utilizando una escala lineal y ordenadas por magnitud, lo que facilita el análisis de las propiedades estadísticas que sustentan el método. La línea discontinua corresponde a un ajuste de ley de potencias del histograma. Esta ley de potencias se ajusta aproximadamente a los datos tanto en la región de frecuencias altas como bajas, lo que permite estimar los parámetros de la ley de potencias empleando únicamente los elementos más frecuentes.

red, una pequeña fracción de los flujos concentra la mayoría de las altas frecuencias, y por lo tanto contribuye significativamente más al valor de la entropía empírica que los flujos menos frecuentes. Esta observación permite definir una variable aleatoria, con la cual se mapean los flujos de red observados en el intervalo a valores enteros consecutivos. Los flujos se ordenaron por probabilidades ($\frac{\hat{m}_i}{M}$) y se les asignaron valores enteros en orden decreciente de frecuencia, lo que resultó en el histograma mostrado en la Figura 4.1. Segundo, la distribución de una fracción significativa de los elementos sigue aproximadamente una ley de potencias, representada por el ajuste lineal en negro (en escala log-log) en la Figura 4.1. La distribución de ley de potencias aparece comúnmente en muchas redes tecnológicas, biológicas y lingüísticas [97]. La distribución empírica muestra una meseta para valores altos de i (aproximadamente $i > 10^6$ en la Figura 4.1), lo cual indica que numerosos flujos aparecen una sola vez en el intervalo de observación. Con base en estas propiedades, se propone almacenar únicamente las frecuencias de los flujos más frecuentes y estimar las frecuencias del resto. De esta manera, el método calcula una estimación

de la entropía empírica al separar la Ecuación (4.3) en tres componentes estimados:

$$\hat{H} = \hat{H}_1 + \hat{H}_2 + \hat{H}_3. \tag{4.5}$$

El primer componente $\hat{H}_1$ resume la contribución de los elementos más frecuentes, cuyas frecuencias reales son estimadas y almacenadas por el algoritmo propuesto, siguiendo un enfoque similar al presentado en [34]. Por lo tanto, se calcula como:

$$\hat{H}_1 = -\sum_{i=1}^{K} \frac{\hat{m}_i}{M} \log \frac{\hat{m}_i}{M}, \tag{4.6}$$

donde K es un parámetro elegido por el algoritmo, M se calcula mediante un contador simple de paquetes, i es el índice de cada uno de los top-K elementos en el histograma, y $\hat{m}_i$ son las frecuencias estimadas por *sketches* de cuentas como los descritos en el capítulo anterior.

El segundo término de la Ecuación (4.5) estima la contribución a la entropía de aquellos flujos que no están dentro de los top-K y que presentan un comportamiento acorde con una ley de potencias. Estos flujos corresponden al rango $K + 1 \leq i \leq L$, donde L es el valor de i en el histograma que representa el primer elemento con frecuencia unitaria, como se observa en la Figura 4.1. La ley de potencias se define como:

$$\hat{p}_i = Ci^{-\alpha}, \tag{4.7}$$

donde C y α son los parámetros de la distribución [98]. Dado que el comportamiento de ley de potencias también puede observarse en la mayoría de los flujos del conjunto top-K, se utilizan esos datos para estimar los valores de C y α en el rango $K + 1 \leq i \leq L$. Por lo tanto, se calcula $\hat{H}_2$ como:

$$\hat{H}_2 = -\sum_{i=K+1}^{\hat{L}} \hat{C}i^{-\hat{\alpha}} \log(\hat{C}i^{-\hat{\alpha}}), \tag{4.8}$$

donde $\hat{C}$ y $\hat{\alpha}$ son los parámetros estimados de la distribución de ley de potencias y $\hat{L}$ es una estimación de L, calculada como el valor de i para el cual se cumple que $\hat{C}i^{-\hat{\alpha}} = \frac{1}{M}$. En la siguiente sección se describe los algoritmos utilizados para calcular $\hat{H}_2$.

El tercer término de la Ecuación (4.5) corresponde a todos aquellos flujos que aparecen una sola vez en el intervalo de observación, es decir, aquellos $\hat{p}_i = 1/M$ para los cuales $L \leq i \leq N$.

Por lo tanto, $\hat{H}_3$ puede expresarse como:

$$\hat{H}_3 = \frac{\hat{N} - \hat{L}}{M} \log M, \tag{4.9}$$

donde $\hat{N}$ es el número de flujos distintos en el intervalo de observación, estimado mediante *sketches* de cardinalidad como el HLL descrito en el capítulo anterior.

Finalmente, calculamos la entropía estimada normalizada como:

$$\hat{H}_{norm} = \frac{\hat{H}}{\log \hat{N}}. \tag{4.10}$$

En resumen, para estimar la entropía normalizada el método requiere seleccionar el número de elementos más frecuentes K, para los cuales se calcularán y almacenarán las estimaciones de frecuencia $\hat{m}_i$, estimar los parámetros de la distribución de ley de potencias $\hat{\alpha}$ y $\hat{C}$; y el índice $\hat{L}$, que representa el índice del primer elemento con frecuencia unitaria en el histograma. Además, se debe estimar el número total de flujos distintos $\hat{N}$. Finalmente, se puede calcular la entropía normalizada utilizando las Ecuaciones (4.5)–(4.10). La siguiente sección describe el algoritmo utilizado para estimar la entropía normalizada empleando el método propuesto.

4.4. Algoritmo

En el Algoritmo 1 se muestran los pasos principales del algoritmo propuesto para la estimación de entropía. Primero, todas las estructuras de datos se inicializan en las líneas 1–5. A continuación, en las líneas 6–10, el algoritmo procesa los paquetes entrantes en modo *streaming*, insertando los identificadores de flujo e de cada paquete en dos *sketches*: *CardSketch*, que estima la cardinalidad $\hat{N}$ del conjunto de datos, y *FreqSketch*, que estima las frecuencias de paquetes $\hat{m}_i$ de cada flujo. Además, el algoritmo incrementa el contador de paquetes M e inserta el identificador de flujo y su frecuencia (estimada por *FreqSketch*) en una cola de prioridad o PQ de tamaño fijo, que almacena los flujos más frecuentes. El tamaño K_{pq} de PQ es un parámetro definido por el usuario. Como se discutió en el capítulo anterior, se elige K_{pq} tal que $K_{pq} \geq K$.

Al finalizar el intervalo de observación el algoritmo termina de procesar el *stream* de paquetes y comienza la estimación de entropía. En las líneas 11–13, lee el contenido de la estructura PQ para (a) calcular K, el número de elementos frecuentes almacenados en la PQ, y (b) calcular $\hat{H}_1$,

Algoritmo 1: Estimación de la entropía empírica normalizada en intervalos de tiempo discretos.

Input: tráfico de red, tamaño de la cola de prioridad K_{pq}, dimensiones del *sketch* de frecuencias d y w, precisión del *sketch* de cardinalidad p, función hash h

Output: entropía normalizada $\hat{H}_{norm}$

Let
- $K \leftarrow 0$, $\hat{L} \leftarrow 0$, $M \leftarrow 0$, $\hat{C} \leftarrow 0$, $\hat{\alpha} \leftarrow 0$, $\hat{H} \leftarrow 0$, $\hat{H}_1 \leftarrow 0$, $\hat{H}_2 \leftarrow 0$, $\hat{H}_3 \leftarrow 0$
- *FreqSketch.inicialización* (d, w, h)
- *CardSketch.inicialización* $(p; h)$
- *PQ.inicialización* (K_{pq})

`// Procesamiento de los flujos de red`

foreach *paquete con identificador de flujo e* ***in*** *stream* **do**
- *FreqSketch.actualización(e)*
- *CardSketch.actualización(e)*
- $M \leftarrow M + 1$
- *PQ.actualización(e, FreqSketch.estimación(e))*

`// Contribución de los elementos más frecuentes`

foreach *frecuencia* m_i ***in*** PQ **do**
- $K \leftarrow K + 1$
- $\hat{H}_1 \leftarrow$ Estimación usando M, m_i y Ecuación (4.6)

`// Cardinalidad del conjunto de datos`

$\hat{N} \leftarrow$ *CardSketch.estimación(e)*

`// Estimación de los parámetros de la distribución de la ley de potencias`

SortedPQ $\leftarrow$ *Sort(PQ)*

$\hat{\alpha} \leftarrow$ Estimación usando K, $\hat{m}_i$, i y Ecuación (4.11)

$\hat{L} \leftarrow$ Estimación usando K, $\hat{m}_K$, $\hat{\alpha}$ y Ecuación (4.12)

`// Entropía estimada`

if $\hat{L} < \hat{N}$ **then**
- $\hat{C} \leftarrow$ Estimación usando M, $\hat{N}$, K, $\hat{m}_i$, $\hat{\alpha}$, $\hat{L}$ y Ecuación (4.16)
- $\hat{H}_2 \leftarrow$ Estimación usando K, $\hat{\alpha}$, $\hat{L}$, $\hat{C}$ y Ecuación (4.20)
- $\hat{H}_3 \leftarrow$ Estimación usando M, $\hat{N}$, $\hat{L}$ y Ecuación (4.9)
- $\hat{H} \leftarrow \hat{H}_1 + \hat{H}_2 + \hat{H}_3$

else
- $\hat{C} \leftarrow$ Estimación usando M, $\hat{N}$, K, $\hat{m}_i$, $\hat{\alpha}$ y Ecuación (4.22)
- $\hat{H}_2 \leftarrow$ Estimación usando K, $\hat{N}$, $\hat{\alpha}$, $\hat{C}$ y Ecuación (4.24)
- $\hat{H} \leftarrow \hat{H}_1 + \hat{H}_2$

`// Estimación y retorno de la entropía normalizada`

$\hat{H}_{norm} \leftarrow$ Estimación usando $\hat{H}$, $\hat{N}$ y Ecuación (4.10)

return $\hat{H}_{norm}$

la contribución a la entropía de los elementos contenidos en la PQ, utilizando la Ecuación (4.6). En la línea 14, el algoritmo emplea la estimación de cardinalidad proporcionada por *CardSketch* para obtener $\hat{N}$, el número estimado de flujos diferentes presentes en el *stream* de datos.

En las líneas 15–27, el algoritmo calcula $\hat{H}_2$ y $\hat{H}_3$, la contribución del resto de los flujos a la entropía empírica estimada. Primero, se calcula una estimación $\hat{\alpha}$ del parámetro exponencial α de la distribución de ley de potencias en la Ecuación (4.7). Existen dos enfoques comunes para estimar $\hat{\alpha}$: el primero utiliza regresión lineal mediante el método de los mínimos cuadrados (*Least squares method, LSM*) [99] sobre el histograma de frecuencias normalizadas con escala logarítmica en ambos ejes. El segundo método emplea un estimador de máxima verosimilitud [100, 101], que es estadísticamente más robusto que el de mínimos cuadrados. Sin embargo, también es computacionalmente más complejo y sus resultados son más sensibles a la precisión con la que se estima el punto a partir del cual las frecuencias comienzan a seguir una ley de potencias. Por esta razón, se utiliza mínimos cuadrados para calcular $\hat{\alpha}$ como la pendiente del ajuste lineal en la gráfica log-log del histograma, utilizando las K frecuencias almacenadas en la PQ:

$$\hat{\alpha} = \frac{\sum_{i=1}^{K}(\log \hat{m}_i \log i) - \frac{(\sum_{i=1}^{K} \log i)(\sum_{i=1}^{K} \log \hat{m}_i)}{\log K}}{\sum_{i=1}^{K}(\log i)^2 - \frac{\left(\sum_{i=1}^{K} \log i\right)^2}{\log K}}. \tag{4.11}$$

La Ecuación (4.11) requiere ordenar las frecuencias en la PQ en orden decreciente para construir el histograma. En el capítulo anterior, se analizó que la estructura de datos utilizada para implementar la PQ no mantiene los datos ordenados; por lo tanto, el primer paso es ordenar los elementos de la PQ, como se muestra en la línea 15. Luego, en la línea 16, se calcula el valor de $\hat{\alpha}$ utilizando la Ecuación (4.11).

En la línea 17, se calcula $\hat{L}$, el valor estimado mínimo de i para el cual $\hat{p}_i = \frac{1}{M}$ en el histograma. Se utilizó una ecuación lineal simple basada en los puntos $(K, \frac{\hat{m}_K}{M})$ y $(\hat{L}, \frac{1}{M})$ en escala logarítmica (log-log):

$$\hat{L} = 2^{\log K - \frac{\log \hat{m}_K}{\hat{\alpha}}}. \tag{4.12}$$

Dado que $L < N$, normalmente el algoritmo generará estimaciones tales que $\hat{L} < \hat{N}$. Primero abordamos este caso común, y al final de la sección discutimos el caso en el que $\hat{L} \geq \hat{N}$.

Las líneas 19–22 estiman los valores de $\hat{C}$, $\hat{H}_2$, $\hat{H}_3$ y $\hat{H}$ cuando $\hat{L} < \hat{N}$. Primero, se calcula

el valor de $\hat{C}$ considerando que la suma de todas las probabilidades debe ser igual a uno:

$$\sum_{i=1}^{\hat{N}} \hat{p}_i = \sum_{i=1}^{K} \frac{\hat{m}_i}{M} + \sum_{i=K+1}^{\hat{L}} \hat{C} i^{-\hat{\alpha}} + \sum_{\hat{L}+1}^{\hat{N}} \frac{1}{M} = 1. \tag{4.13}$$

La Ecuación (4.13) se usa para normalizar la suma de las frecuencias en la estimación y calcular el valor de $\hat{C}$ como:

$$\hat{C} = \frac{1 - \frac{\hat{N}-\hat{L}}{M} - \sum_{i=1}^{K} \frac{\hat{m}_i}{M}}{\sum_{i=K+1}^{\hat{L}} i^{-\hat{\alpha}}}. \tag{4.14}$$

La sumatoria $\sum_{i=1}^{K} \frac{\hat{m}_i}{M}$ puede calcularse al mismo tiempo que $\hat{H}_1$ y consiste en solo K términos, lo cual es relativamente pequeño. Sin embargo, $\sum_{i=K+1}^{\hat{L}} i^{-\hat{\alpha}}$ es computacionalmente más costosa, ya que requiere una potenciación para cada uno de sus $\hat{L} - \hat{K}$ términos, y normalmente $\hat{L} - \hat{K} \gg \hat{K}$. Por lo tanto, se utiliza el formalismo continuo de la ley de potencias, que se emplea frecuentemente para simplificar cálculos discretos [97]. Suponiendo que i es una variable aleatoria continua con una función de densidad de probabilidad $p(i) = \hat{C} i^{-\hat{\alpha}}$, se aproxima la sumatoria $\sum_{i=K+1}^{\hat{L}} i^{-\hat{\alpha}}$ como:

$$\sum_{i=K+1}^{\hat{L}} i^{-\hat{\alpha}} \approx \int_{i=K+1}^{i=\hat{L}} i^{-\hat{\alpha}} di = \frac{\hat{L}^{1-\hat{\alpha}} - (K+1)^{1-\hat{\alpha}}}{1-\hat{\alpha}} \equiv I_{\hat{L}}. \tag{4.15}$$

Así, usando la Ecuación (4.14), se estima C en la línea 19 como:

$$\hat{C} = \frac{(1-\hat{\alpha})(1 - \frac{\hat{N}-\hat{L}}{M} - \sum_{i=1}^{K} \frac{\hat{m}_i}{M})}{\hat{L}^{1-\hat{\alpha}} - (K+1)^{1-\hat{\alpha}}}. \tag{4.16}$$

A continuación, se calcula el valor de $\hat{H}_2$. Primero, se usan las reglas de los logaritmos y la propiedad $\ln i = \frac{\log i}{\log e}$ para reemplazar la Ecuación (4.8) por:

$$\hat{H}_2 = \hat{C} \log \hat{C} \sum_{i=K+1}^{\hat{L}} i^{-\hat{\alpha}} - \hat{C}\hat{\alpha} \log e \sum_{i=K+1}^{\hat{L}} i^{-\hat{\alpha}} \ln i. \tag{4.17}$$

Se aproximó $\sum_{i=K+1}^{\hat{L}} i^{-\hat{\alpha}}$ con $I_{\hat{L}}$ tal como se define en la Ecuación (4.15). Para aproximar $\sum_{i=K+1}^{\hat{L}} i^{-\hat{\alpha}} \ln i$, se usa $i^{-\alpha} \ln i = \frac{\partial}{\partial \alpha} i^{-\alpha}$. Dado que la suma de derivadas es igual a la derivada

de la suma, se obtiene:

$$\sum_{i=K+1}^{\hat{L}} i^{-\hat{\alpha}} \ln i = \frac{\partial}{\partial \hat{\alpha}} \sum_{i=K+1}^{\hat{L}} i^{-\hat{\alpha}} \approx \frac{\partial}{\partial \hat{\alpha}} \left(\frac{i^{1-\hat{\alpha}}}{1-\hat{\alpha}} \right) \Bigg|_{(K+1)}^{\hat{L}}. \tag{4.18}$$

Resolviendo la derivada, se obtiene:

$$\frac{\partial}{\partial \hat{\alpha}} \left(\frac{i^{1-\hat{\alpha}}}{1-\hat{\alpha}} \right) \Bigg|_{(K+1)}^{\hat{L}} = \left(- \frac{\ln i}{(\hat{\alpha}-1) i^{\hat{\alpha}-1}} - \frac{1}{(\hat{\alpha}-1)^2 i^{\hat{\alpha}-1}} \right) \Bigg|_{(K+1)}^{\hat{L}} \equiv D_{\hat{L}}. \tag{4.19}$$

Reemplazando las Ecuaciones (4.15), (4.16) y (4.19) en la Ecuación (4.17), se calculó $\hat{H}_2$ en la línea 20 como:

$$\hat{H}_2 = \left(1 - \frac{\hat{N} - \hat{L}}{M} - \sum_{i=1}^{K} \frac{\hat{m}_i}{M} \right) \log \hat{C} - \hat{C} \hat{\alpha} D_{\hat{L}} \log e. \tag{4.20}$$

Después de calcular $\hat{H}_2$, el algoritmo calcula $\hat{H}_3$ en la línea 21 usando la Ecuación (4.9), y luego $\hat{H}$ en la línea 22 como $\hat{H} = \hat{H}_1 + \hat{H}_2 + \hat{H}_3$.

En las líneas 24 a la 26 del algoritmo abordan el caso en que $\hat{L} \geq \hat{N}$, lo cual es poco frecuente pero puede ocurrir cuando los parámetros de la ley de potencias se desvían significativamente de la distribución empírica de probabilidad del conjunto de datos. En este caso, se ignoran las frecuencias estimadas de los flujos más allá de $\hat{N}$, es decir, se usa $\hat{L} = \hat{N}$. Así, $\hat{H}_3 = 0$ y el cálculo de $\hat{C}$ se reduce a:

$$\hat{C} = \frac{1 - \sum_{i=1}^{K} \frac{\hat{m}_i}{M}}{\sum_{i=K+1}^{\hat{N}} i^{-\hat{\alpha}}}. \tag{4.21}$$

Usando la Ecuación (4.15) y $\hat{L} = \hat{N}$, se calcula $\hat{C}$ en la línea 24 como:

$$\hat{C} = \frac{(-\hat{\alpha}+1)(1 - \sum_{i=1}^{K} \frac{\hat{m}_i}{M})}{\hat{N}^{-\hat{\alpha}+1} - (K+1)^{-\hat{\alpha}+1}}. \tag{4.22}$$

Para estimar $\hat{H}_2$, se define $D_{\hat{N}}$ usando la Ecuación (4.19) con $\hat{L} = \hat{N}$:

$$\frac{\partial}{\partial \hat{\alpha}} \left(\frac{i^{1-\hat{\alpha}}}{1-\hat{\alpha}} \right) \Bigg|_{(K+1)}^{\hat{N}} = \left(- \frac{\ln i}{(\hat{\alpha}-1) i^{\hat{\alpha}-1}} - \frac{1}{(\hat{\alpha}-1)^2 i^{\hat{\alpha}-1}} \right) \Bigg|_{(K+1)}^{\hat{N}} \equiv D_{\hat{N}}, \tag{4.23}$$

y se estima $\hat{H}_2$ en la línea 25 como:

$$\hat{H}_2 = \left(1 - \sum_{i=1}^{K} \frac{\hat{m}_i}{M} \right) \log \hat{C} - \hat{C} \hat{\alpha} D_{\hat{N}} \log e. \tag{4.24}$$

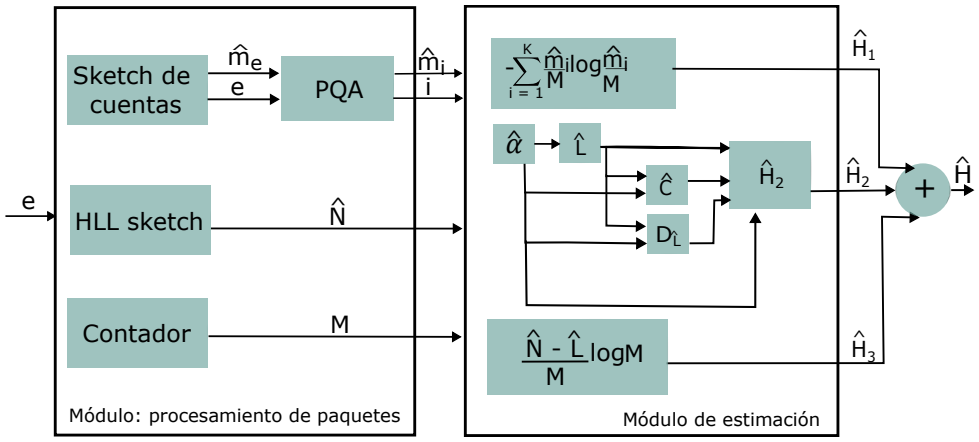


Figura 4.2: Arquitectura del acelerador hardware para la estimación de entropía en intervalos de tiempo discreto.

En la línea 26 se calcula $\hat{H} = \hat{H}_1 + \hat{H}_2$ cuando $\hat{L} \geq \hat{N}$. Finalmente, el algoritmo calcula $\hat{H}_{norm}$ en la línea 27 y devuelve su valor.

4.5. Arquitectura en FPGA

En la Figura 4.2 se muestra la arquitectura del acelerador de estimación de entropía, el cual está diseñado para dispositivos FPGA. Está compuesto por dos módulos principales: el `módulo de procesamiento de paquetes`, que implementa las líneas 6–10 del Algoritmo 1 para estimar las frecuencias de los top-K, la cardinalidad de los flujos y el número de paquetes, y el `módulo de estimación`, que implementa las líneas 11–27 para estimar la entropía a partir de los datos recolectados por el primer módulo.

4.5.1. Módulo de procesamiento de paquetes en tiempo discreto

La implementación del *sketch* de cuentas sigue la misma estructura de la Figura 3.2. Está compuesto por d bloques de memoria de w palabras de 32 bits, que se acceden simultáneamente utilizando d bloques paralelos de la función *MurmurHash3* [93] con diferentes semillas. Como se argumentará en la Sección 4.6.1, un CS proporciona una estimación precisa, por lo tanto se usa esta estructura en el módulo de procesamiento de paquetes. Un bit separado de la misma función *hash* de 32 bits se usa para determinar si se incrementan o decrementan las d celdas seleccionadas en cada acceso, y una red de ordenamiento de profundidad mínima [102] calcula

la mediana de las celdas para entregar la estimación de frecuencia. Las funciones *hash*, el acceso a memoria y la red de ordenamiento se paralelizaron para maximizar la frecuencia de reloj.

La implementación del *sketch* HLL sigue la estructura de la Figura 3.1. Reutiliza un bloque *MurmurHash3* del CS y utiliza un bloque de memoria para implementar el almacenamiento del *sketch*, un bloque que calcula el número de ceros a la izquierda de los b bits del valor *hash* en tiempo logarítmico y un comparador. La implementación está paralelizada para procesar un nuevo identificador de flujo en cada ciclo de reloj. La PQA utiliza S bloques de memoria, accedidos en paralelo mediante un segundo bloque *MurmurHash3* compartido con el CS. En cada acceso a la PQA, las celdas seleccionadas se tratan como una cola de prioridad de S elementos, y su contenido se actualiza siguiendo el procedimiento descrito en el capítulo anterior. Para actualizar la cola, el módulo utiliza un arreglo de S comparadores en paralelo para determinar si el identificador de flujo entrante ya está en la cola o si es necesario insertar un nuevo elemento. El retardo lógico necesario para realizar una operación de actualización es logarítmico en el valor de S, pero para valores pequeños ($S \leq 16$), puede realizarse en un solo ciclo de reloj en el FPGA.

4.5.2. Módulo de estimación de entropía

El cálculo de $\hat{H}_1$ en las líneas 12–13 del Algoritmo 1 requiere leer todos los elementos del PQA. El acelerador reduce el tiempo de ejecución leyendo simultáneamente los S bloques de memoria del PQA y computa los logaritmos y la sumatoria de la Ecuación (4.6) en paralelo. Dado que las FPGAs solo tienen soporte por hardware para sumas y multiplicaciones, el acelerador usa el módulo mostrado en la Figura 4.3 para aproximar la función logaritmo. Este módulo utiliza una tabla de búsqueda (*Lookup Table*, LUT) de 32K entradas y una interpolación lineal para calcular el logaritmo utilizando los 15 bits más significativos de la entrada. Para mejorar la precisión de la aproximación, la entrada se normaliza primero desplazándola a la izquierda hasta que el bit más significativo sea un 1. De este modo, para una entrada entera de n bits, la tabla almacena los valores del logaritmo para entradas en el rango $[2^{n-1}, 2^n[$. El número de desplazamientos de bits usados para normalizar la entrada se resta del resultado para compensar la normalización. El bloque `ldz` que computa el número de ceros a la izquierda es el mismo que se usa en la implementación del *sketch* HLL [34]. Módulos similares de logaritmo también se utilizan en el cálculo de $\hat{H}_2$ y $\hat{H}_3$. En algunos casos, por ejemplo al calcular $\log \hat{C}$ en la Ecuación (4.20), la entrada está codificada como un número de punto fijo, por lo que el número de bits fraccionales se resta del resultado para compensar que el módulo trata la entrada como un número entero. Como los bloques de memoria en FPGA son de doble puerto, la misma LUT es compartida por

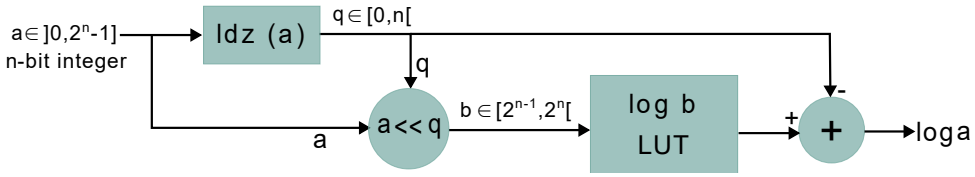


Figura 4.3: Módulo de cálculo del logaritmo en base 2.

dos módulos de logaritmo.

El cálculo de $\hat{H}_2$ contiene la operación más costosa en tiempo del algoritmo, es decir, ordenar los elementos del PQA en la línea 15 del Algoritmo 1 para calcular $\hat{\alpha}$ mediante el método de mínimos cuadrados. Se utilizó el algoritmo de ordenamiento *radix-sort* [103] con un *radix* de 4 bits, el cual ordena un arreglo de números en $n/4$ iteraciones, donde n es el número de bits usados para representar cada número. En cada iteración, el algoritmo ordena los números utilizando un segmento de 4 bits adyacentes. Dado que se usaron 32 bits para representar las frecuencias de los flujos, el algoritmo ordena el PQA en ocho iteraciones. En la Figura 4.4 se ilustra el funcionamiento del algoritmo para un ejemplo pequeño de seis números de 4 bits, usando un *radix* de 2 bits. Los dos bits usados para ordenar la entrada en cada iteración están resaltados en verde en la figura. En este caso, el algoritmo ordena el arreglo en dos iteraciones, etiquetadas como `Paso 1` y `Paso 2` en la figura. Dado que cada iteración requiere ordenar el contenido del arreglo hacia un segundo arreglo, la implementación realizada utiliza un doble búfer que alterna los roles de entrada/salida en cada iteración (`Buffer 0/1` en la figura). También se usa una tabla de 16 palabras (4 palabras en el ejemplo con *radix* de 2 bits en la Figura 4.4), que se utiliza para calcular los índices desde los cuales los números deben escribirse en el búfer de salida durante cada iteración. Esto se realiza en dos pasos: primero el algoritmo lee el búfer de entrada y computa un histograma con un intervalo para cada valor de *radix* (`Tabla 0`). Luego, calcula el histograma acumulativo para determinar el índice inicial de cada valor de *radix* (`Tabla 1`). Finalmente, copia el contenido del búfer de entrada al búfer de salida utilizando estos índices.

Aunque el algoritmo *radix-sort* se ejecuta en tiempo lineal (para un número fijo de bits), sigue siendo la operación más costosa en términos de tiempo dentro de la etapa de estimación del algoritmo. Para reducir la latencia, el acelerador aplica dos aproximaciones:

1. En lugar de ordenar globalmente el contenido del PQA, el acelerador ordena sus S bloques de memoria de forma independiente y en paralelo, y luego los concatena. Debido a que las frecuencias están distribuidas uniformemente en el PQA, el arreglo resultante, aunque no esté perfectamente ordenado, es una buena aproximación de un PQA ordenado.

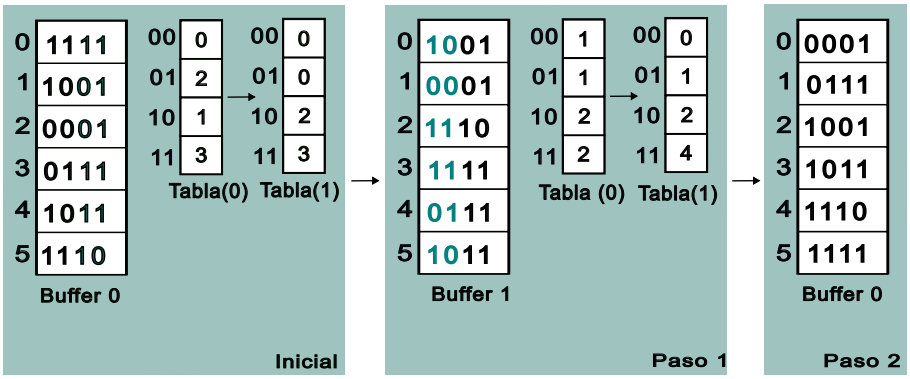


Figura 4.4: Ejecución del algoritmo de ordenamiento *radix-sort* en el acelerador.

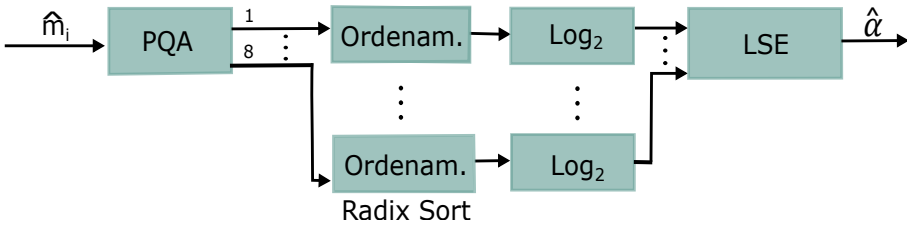


Figura 4.5: Módulo de estimación de $\hat{\alpha}$.

2. En lugar de ordenar los R elementos de cada bloque, el acelerador ordena solo una fracción de ellos ($R/4$ en nuestra implementación). Dado que la función *hash* distribuye uniformemente las frecuencias entre las R colas del PQA, esto es equivalente a realizar un muestreo uniforme del contenido del PQA.

Usando las aproximaciones descritas anteriormente, el tiempo de ordenamiento se acelera por un factor de $4S$, y aun así se logra una buena aproximación de $\hat{\alpha}$ y $\hat{H}_2$. El proceso de ordenar los bloques de memoria que componen el PQA de forma independiente y en paralelo, en lugar de como un solo bloque, para el cálculo de $\hat{\alpha}$ se muestra en la Figura 4.5.

La última función aritmética no trivial requerida por el acelerador es a^b, que se utiliza, por ejemplo, en las Ecuaciones.(4.19),(4.16) y (4.22). Usar una LUT para calcular esta función es poco práctico porque tiene dos entradas. Afortunadamente, se puede usar la propiedad $a^b = 2^{b \log a}$ para expresar el cálculo como dos funciones de una sola entrada y una multiplicación. La implementación en el acelerador, mostrada en la Figura 4.6, utiliza una instancia del módulo de logaritmo descrito anteriormente y una segunda LUT para calcular 2^y.

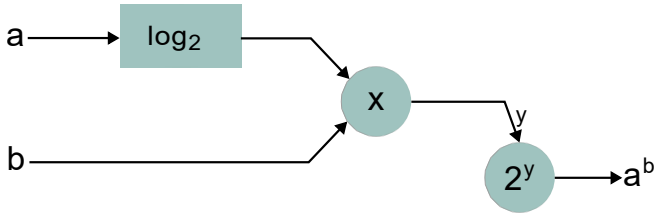


Figura 4.6: Módulo de cálculo de potencias.

4.6. Resultados

En los experimentos realizados se usaron 16 trazas de conjuntos de datos públicos: seis del Centro para el Análisis Aplicado de Datos de Internet (*Center for Applied Internet Data Analysis*, CAIDA) [32], seis de MawiLab [33], y cuatro del conjunto de datos etiquetado IoT-23 [96]. Los conjuntos de datos de CAIDA y MawiLab contienen trazas recolectadas desde monitores de alta velocidad en enlaces troncales comerciales. Las trazas de MawiLab presentan varias anomalías de red, como ataques de inundación SYN, RST, SMB, Ping, entre otros. Las anomalías fueron etiquetadas usando detectores de anomalías [33, 104, 105, 106, 107], y están documentadas en los repositorios de MawiLab. Las trazas de CAIDA no están etiquetadas. También se incluyeron trazas del conjunto de datos IoT-23, que consiste en tráfico de red proveniente de dispositivos del Internet de las Cosas. Combinan tráfico normal y tráfico con características de botnets comúnmente asociadas con ataques DDoS. Específicamente, Capture-7-1 y Capture-35-1 contienen trazas de la botnet Mirai, Capture-17-1 contiene trazas de Kenjiro, y Capture-60-1 contiene trazas de Gagfyt. En la Tabla 4.1 se resumen las trazas, incluyendo los valores exactos del número total de paquetes M, la cardinalidad de flujos N, y su entropía normalizada utilizando la tupla completa de cinco atributos del flujo: dirección origen y destino, puerto origen y destino, y protocolo. Un pequeño número de paquetes tienen uno o más atributos vacíos, en cuyo caso el algoritmo los trata como redundantes y los ignora [108]. Las trazas contienen entre 65 mil y 50 millones de flujos, y entre 4 millones y 271 millones de paquetes.

Para realizar la evaluación, primero se selecciona el número de frecuencias estimadas empíricamente K, el parámetro p en el *sketch* HLL, el tipo y las dimensiones del *sketch* de cuentas, y las dimensiones del PQA. En la Sección 4.6.1, se utilizan dos trazas seleccionadas aleatoriamente de cada conjunto de datos para elegir parámetros que funcionen bien en escenarios del mundo real. Luego, en las Secciones 4.6.2 y 4.6.3 se evaluan los resultados del método, el algoritmo y el acelerador propuesto usando las 16 trazas, y se compara su error de estimación y eficiencia en espacio/tiempo con trabajos anteriores.

Tabla 4.1: Trazas de tráfico de red utilizadas en los experimentos, donde M, N y H_{norm} representan el número total de paquetes, la cardinalidad de los flujos y la entropía normalizada exacta, respectivamente.

Trazas	M	N	H_{norm}
Chicago-20150219	15,808,577	635,775	0.7546
Chicago-20160121	31,166,491	866,722	0.7373
Chicago-20110608	19,428,464	1,572,543	0.7388
Sanjose-20081016	20,401,235	1,804,149	0.7880
Chicago-20080515	11,892,587	937,839	0.7969
Chicago-20080319	3,895,536	395,055	0.8419
Mawi-20211400	37,625,618	12,350,968	0.7434
Mawi-20201400	119,874,474	44,643,492	0.7133
Mawi-20191400	62,477,881	31,943,925	0.7844
Mawi-20181400	76,006,586	26,899,558	0.6925
Mawi-20171400	115,483,340	23,298,270	0.6746
Mawi-20161400	94,728,786	26,749,003	0.7063
Capture-7-1	11,500,821	11,380,845	0.9965
Capture-17-1	109,380,499	50,210,856	0.9982
Capture-35-1	46,825,584	4,378,455	0.6566
Capture-60-1	271,138,264	65,773	0.2578

4.6.1. Parámetros del algoritmo

Número de elementos Top-K

En los experimentos se usa el error relativo de estimación para evaluar el impacto del valor de K en el rendimiento del método propuesto en la Sección 4.3. El error se define como:

$$E_{r\_PL} = \frac{|H_{norm} - \hat{H}_{norm\_PL}|}{H_{norm}} \times 100, \tag{4.25}$$

donde H_{norm} es la entropía normalizada exacta de la traza, según lo definido por la Ecuación (4.4). $\hat{H}_{norm\_PL}$ es la estimación del método, tal como se indica en la Ecuación (4.10), pero utilizando los valores exactos de cardinalidad N y de las frecuencias m_i de los elementos top-K.

En la Figura 4.7 se muestra el error de estimación en las seis trazas seleccionadas, en función de K, expresado en potencias de dos entre 4,096 y 32,768. La estimación utiliza el método

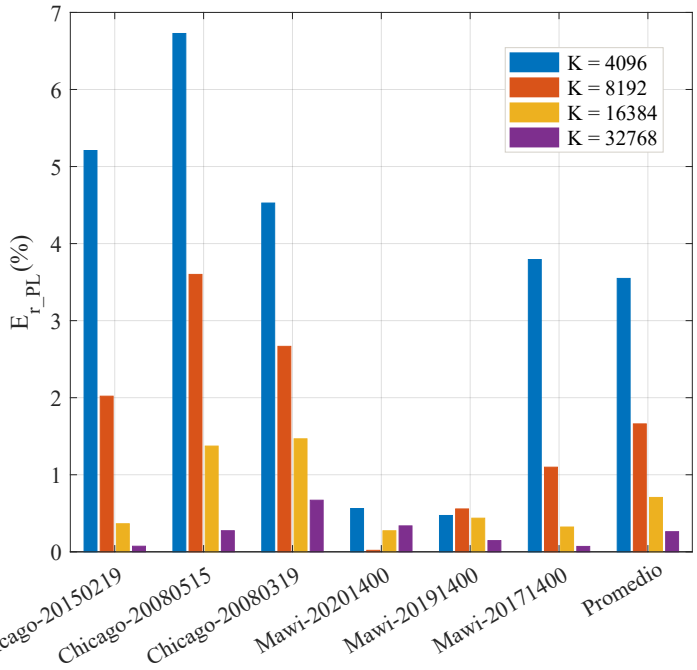


Figura 4.7: Error de estimación de entropía del método propuesto relativo al valor exacto de la entropía calculada con la Ecuación (4.4), en función de K.

propuesto en la Sección 4.3 con los valores exactos de N y m_i. El último grupo de barras muestra el promedio del error relativo de las seis trazas. El error disminuye para valores mayores de K, ya que en esos casos se pueden estimar mejor los parámetros de la distribución de ley de potencias que se usa para calcular $\hat{H}_2$. Para $K \geq 16,384$, el valor medio de E_{r_PL} es menor al 0.9 % y su valor máximo es inferior al 2.9 %, lo cual ha demostrado ser suficiente para discriminar entre la entropía de tráfico normal y distintos tipos de ataques DoS y DDoS [18]. Por lo tanto, se selecciona $K = 16,384$ como un buen compromiso entre el error de estimación y el uso de memoria en un *switch* programable o un acelerador hardware basado en FPGA [34].

Parámetro de precisión del *sketch* HLL

Para determinar el parámetro de precisión p en el *sketch* HLL se evalúa el impacto del error de estimación de cardinalidad sobre la entropía estimada. El error relativo de estimación causado por el *sketch* HLL se define como:

$$E_{r\_HLL} = \frac{|\hat{H}_{norm\_PL} - \hat{H}_{norm\_HLL}|}{\hat{H}_{norm\_PL}} \times 100 \tag{4.26}$$

donde $\hat{H}_{norm\_HLL}$ es la entropía normalizada calculada mediante la Ecuación (4.10) con $K =$

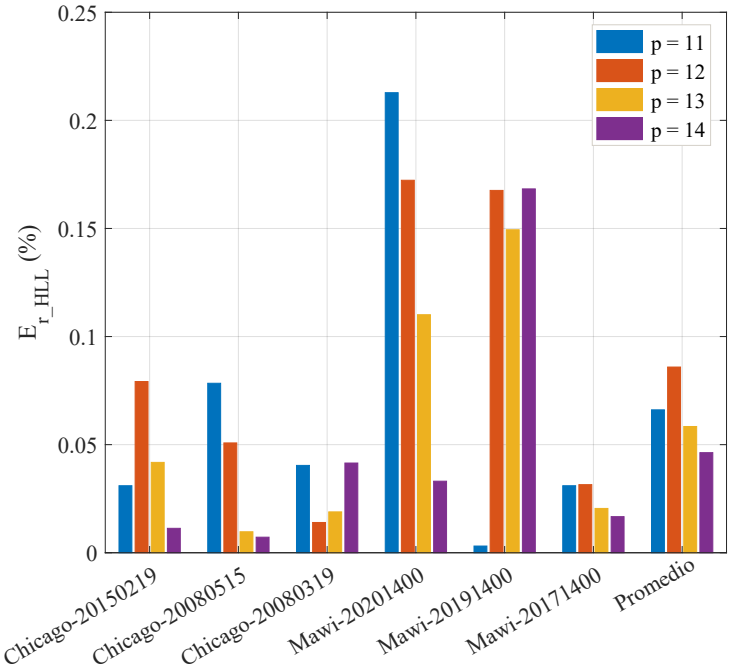


Figura 4.8: Error de estimación de entropía del algoritmo relativo a la entropía estimada mediante la Ecuación. (4.10), en función del parámetro de precisión p del HLL.

$16,384$, utilizando el valor de $\hat{N}$ estimado por el *sketch* HLL. La implementación de HLL utiliza la función *hash MurmurHash3* de 32 bits [93]. La Figura 4.8 muestra que para $p \geq 13$, $E_{r\_HLL} < 0{,}12\,\%$ en las seis trazas, con un valor medio inferior al $0.03\,\%$. Por lo tanto, seleccionamos $p = 13$ para implementar el *sketch* HLL en el algoritmo.

Sketch de cuentas: tipo y dimensiones

En el tercer experimento se evalúa el impacto del *sketch* de cuentas en la estimación de las frecuencias. Se usa este análisis para seleccionar el tipo de *sketch* y sus dimensiones. Se consideraron dos *sketches*, CM-CU y CS, debido a su buen desempeño teórico y empírico en cuanto al error de estimación [27, 109, 110, 34, 111]. También se evaluó un conjunto de tamaños de *sketch*, con $w \in \{32768, 65536\}$ y $d \in \{9, 11, 13\}$. Se evaluó el desempeño del *sketch* utilizando el error de estimación de entropía relativo a la estimación del método $\hat{H}_{norm\_PL}$:

$$E_{r\_s} = \frac{|\hat{H}_{norm\_PL} - \hat{H}_{norm\_s}|}{\hat{H}_{norm\_PL}} \times 100 \tag{4.27}$$

donde $\hat{H}_{norm\_s}$ es la entropía normalizada estimada mediante la Ecuación (4.10), utilizando las frecuencias estimadas $\hat{m}_i$ por el *sketch* de cuentas y la cardinalidad $\hat{N}$ producida por el *sketch*

Tabla 4.2: Error de estimación del algoritmo para dos *sketches* de cuentas: CS y CM-CU y seis dimensiones diferentes.

Trazas: $\hat{H}_{norm\_PL}$	w	$E_{r\_s}(\%)$					
		CS			CM-CU		
		$d=9$	$d=11$	$d=13$	$d=9$	$d=11$	$d=13$
Chicago-20150219	32K	0.3759	0.1920	0.1223	0.1503	0.1370	0.1280
$\hat{H}_{norm\_PL}=0{,}7573$	64K	0.6722	**0.0495**	0.0464	0.0720	0.0677	0.0648
Chicago-20080515	32K	0.2848	0.2042	0.1206	0.1073	0.0934	0.0841
$\hat{H}_{norm\_PL}=0{,}8079$	64K	0.6625	**0.0148**	0.0058	0.0250	0.0208	0.0180
Mawi-20201400	32K	0.1479	1.4896	1.9559	1.5282	1.3295	1.1947
$\hat{H}_{norm\_PL}=0{,}7114$	64K	3.5665	**0.0652**	0.3890	0.3262	0.2878	0.2596
Mawi-20171400	32K	8.9697	2.5186	1.7539	1.2423	1.0972	1.0206
$\hat{H}_{norm\_PL}=0{,}6768$	64K	0.9933	**0.0742**	0.4889	0.4361	0.4027	0.3786
Capture-17-1	32K	0.2232	0.1897	0.1622	5.0757	4.7132	4.4395
$\hat{H}_{norm\_PL}=0{,}9991$	64K	0.1177	**0.0953**	0.0785	2.1434	1.9835	1.8615
Capture-35-1	32K	0.8794	0.7497	0.6542	2.2760	2.0680	1.9203
$\hat{H}_{norm\_PL}=0{,}6753$	64K	0.4853	**0.4130**	0.3601	0.8934	0.8180	0.7611

HLL con $p=13$ y $K=16,384$.

En la Tabla 4.2 se muestran los resultados obtenidos. Cada fila de la tabla presenta el E_{r_s} para cada uno de las seis trazas seleccionadas, considerando los distintos tipos y tamaños de *sketch*. En la tabla se evidencia que el error estimado producido por CS es más sensible al valor de d que CM-CU, mientras que CM-CU es más sensible al valor de w. Se resaltaron los resultados obtenidos con un CS de $65,536 \times 11$, el cual representa el mejor compromiso entre tamaño del *sketch* y error de estimación. Se utilizó este *sketch* y estas dimensiones en la implementación, la cual produce un error medio del 0.12 % relativo a la estimación de entropía dada por la Ecuación (4.10). En comparación con CM-CU, CS requiere una función *hash* adicional por cada una de sus d filas. Sin embargo, como se discute en la Sección 4.5, se puede utilizar una única función *hash* de 32 bits tanto para direccionar la celda como para seleccionar la operación de incremento o decremento. Además, al no usar CM-CU se simplifica la implementación del *sketch* en hardware, ya que el cálculo del valor mínimo durante la operación de inserción introduce una dependencia de múltiples ciclos en arquitecturas profundamente paralelizadas, algo que no ocurre con CS [34].

Dimensiones del PQA

Como se discutió en la Sección 3.4, se reemplazó una cola de prioridad convencional de K elementos por un arreglo de R colas de prioridad de S elementos cada una, tal que $R \times S = K$ y $S \ll K$. Con valores mayores de S, el PQA se aproxima mejor al comportamiento de una cola de prioridad real y mejora el error de estimación, pero la operación de actualización es más lenta y compleja [34]. En esta sección, se consideran dimensiones del PQA de 1024×16, 2048×8, 4096×4 y 8192×2. Se evaluaron estas alternativas utilizando el error de estimación de entropía relativo a $\hat{H}_{norm\_PL}$ con $K = 16,384$, según lo definido en la Ecuación (4.28):

$$E_{r\_PQA} = \frac{|\hat{H}_{norm\_PL} - \hat{H}_{norm\_PQA}|}{\hat{H}_{norm\_PL}} \times 100 \tag{4.28}$$

donde $\hat{H}_{norm\_PQA}$ es la entropía normalizada estimada mediante la Ecuación (4.10), usando un PQA en lugar de una cola de prioridad real de 16,384 elementos, las frecuencias estimadas $\hat{m}_i$ por CS con $w = 65,536$ y $d = 11$, y la cardinalidad estimada $\hat{N}$ por HLL con $p = 13$.

En la Tabla 4.3 se muestra el E_{r_PQA} para las seis trazas y las cuatro configuraciones de dimensiones del PQA. El error de estimación es generalmente más bajo cuando $S = 16$. Sin embargo, con $S = 8$, el error no aumenta significativamente, alcanzando un valor máximo de $0.41\,\%$ y un valor medio inferior al $0.17\,\%$ en relación con la estimación del método dada por la Ecuación (4.10). Además, se ha demostrado que una estructura de datos con estas dimensiones puede implementarse de forma eficiente y con actualizaciones rápidas en un acelerador basado en FPGA [34]. Por lo tanto, en la implementación se utiliza un PQA de 2048 colas de prioridad de 8 elementos cada una.

4.6.2. Resultados del algoritmo

Error de estimación

En esta sección se examina la precisión de estimación del método propuesto y del algoritmo de *streaming* utilizando las 16 trazas de la Tabla 4.1. El algoritmo se implementó usando los parámetros seleccionados en la Sección 4.6.1: $K = 16,384$, HLL con $p = 13$, CS con $w = 65,536$ y $d = 11$, y el PQA con $R = 2,048$ y $S = 8$.

En la Tabla 4.4 se resumen los resultados de estimación de la entropía normalizada. H_{norm} es la entropía exacta de la traza. $\hat{H}_{norm\_PL}$ es la estimación de la entropía calculada mediante

Tabla 4.3: Error de estimación de entropía del algoritmo en función de las dimensiones del PQA, relativo a la estimación del método de la Ecuación (4.10).

Trazas	$E_{r\_PQA}(\%)$			
	1024×16	2048×8	4096×4	8192×2
Chicago-20150219	0.1727	0.3577	0.5832	0.9592
Chicago-20080515	0.0038	0.0758	0.5259	1.2174
Mawi-20201400	0.0899	0.0375	0.0535	0.1718
Mawi-20171400	0.0784	0.0382	0.0247	0.1850
Capture-17-1	0.0964	0.0973	0.0988	0.1015
Capture-35-1	0.4123	0.4112	0.4087	0.4039
Promedio	**0.1423**	**0.1696**	**0.2824**	**0.5065**

el método de la Ecuación (4.10) con $K = 16,384$. $\hat{H}_{norm\_alg}$ es la estimación de la entropía calculada mediante el Algoritmo 1, el cual incluye las aproximaciones de los *sketches* y el PQA en $\hat{H}_{norm\_PQA}$, además de las aproximaciones de la integral y la derivada discutidas en la Sección 4.4. A pesar de todas las aproximaciones introducidas por el algoritmo para estimar la entropía normalizada de los flujos en modo *streaming* y con bajo uso de memoria, los resultados coinciden con las estimaciones del método propuesto hasta el segundo dígito decimal. Aunque no se muestra en la tabla, el error introducido por las aproximaciones de la integral y la derivada respecto a E_{r_PQA} es cero hasta el cuarto dígito decimal.

En la Tabla 4.5 se muestran los errores absoluto y relativo del método propuesto (E_{abs_PL} y E_{r_PL}) y del Algoritmo 1 (E_{abs_alg} y E_{r_alg}), con respecto a la entropía empírica normalizada exacta. El valor medio del error absoluto es menor a 0.006 tanto para el método como para el algoritmo, con valores máximos por debajo de 0.019 y 0.016, y desviaciones estándar de 0.0051 y 0.0047, respectivamente. El error relativo medio es menor al 0.75 % para el método y menor al 0.69 % para el algoritmo, con valores máximos de 2.8 % y 2.4 %, y desviaciones estándar de 0.73 % y 0.66 %, respectivamente. El error medio y máximo de estimación, tanto del método como del algoritmo, están muy por debajo del límite del 3 % que ha sido establecido como un umbral adecuado para discriminar entre tráfico de red normal y anómalo [35, 11]. A pesar de las aproximaciones realizadas por las estructuras de datos utilizadas para reducir el uso de memoria y el tiempo de inserción, el algoritmo produce resultados que se acercan mucho a los del método propuesto. De hecho, para algunas trazas, las aproximaciones empleadas por el algoritmo reducen ligeramente el error de estimación.

En la Figura 4.9 se compara el histograma obtenido a partir de los datos reales (en rojo) con

Tabla 4.4: Entropía exacta calculada con la Ecuación (4.4), junto con las estimaciones del método y el algoritmo propuesto, para las 16 trazas de la Tabla 4.1.

Trazas	H_{norm}	$\hat{H}_{norm\_PL}$	$\hat{H}_{norm\_alg}$
Chicago-20150219	0.7546	0.7573	0.7546
Chicago-20160121	0.7373	0.7464	0.7469
Chicago-20110608	0.7388	0.7416	0.7414
Sanjose-20081016	0.7880	0.7949	0.7915
Chicago-20080515	0.7969	0.8079	0.8073
Chicago-20080319	0.8419	0.8543	0.8539
Mawi-20211400	0.7434	0.7503	0.7499
Mawi-20201400	0.7133	0.7114	0.7117
Mawi-20191400	0.7844	0.7810	0.7822
Mawi-20181400	0.6925	0.6880	0.6879
Mawi-20171400	0.6746	0.6768	0.6771
Mawi-20161400	0.7063	0.7012	0.6990
Capture-7-1	0.9965	0.9965	0.9938
Capture-17-1	0.9982	0.9991	0.9981
Capture-35-1	0.6566	0.6753	0.6725
Capture-60-1	0.2578	0.2578	0.2579

el obtenido por el algoritmo descrito en la Sección 4.4 (en azul), en una escala log-log, para seis trazas. Para los flujos numerados entre 1 y K, la curva azul representa las frecuencias top-K estimadas por el CS y la estructura del PQA divididas por el número de paquetes M. En la mayoría de los casos, el algoritmo sigue de cerca los datos reales, con algunas subestimaciones o sobreestimaciones introducidas por colisiones en el *sketch*. Esto es notorio en la traza Capture-35-1, que presenta el mayor error de estimación. La pequeña caída en la curva cerca de K en las primeras cuatro trazas es producido por el PQA, que omite algunos flujos cercanos al extremo inferior de la cola. Entre los flujos K y $\hat{L}$, el algoritmo estima las frecuencias utilizando un ajuste por ley de potencias. Las gráficas en la Figura 4.9 validan esta suposición, incluso cuando los datos empíricos contienen tráfico anómalo. Las excepciones más notables son las trazas Capture-35-1 y Capture-60-1. Sin embargo, incluso en estas trazas, el error de estimación producido por el algoritmo es inferior al 2.5 % y 0.04 %, respectivamente. Esto se debe a que el algoritmo calcula la entropía utilizando estimaciones empíricas de frecuencia para los flujos top-K y solo utiliza el ajuste por ley de potencias para los flujos menos frecuentes, que tienen una menor contribución a la entropía. Bajo estas condiciones, la suposición de ley de potencias

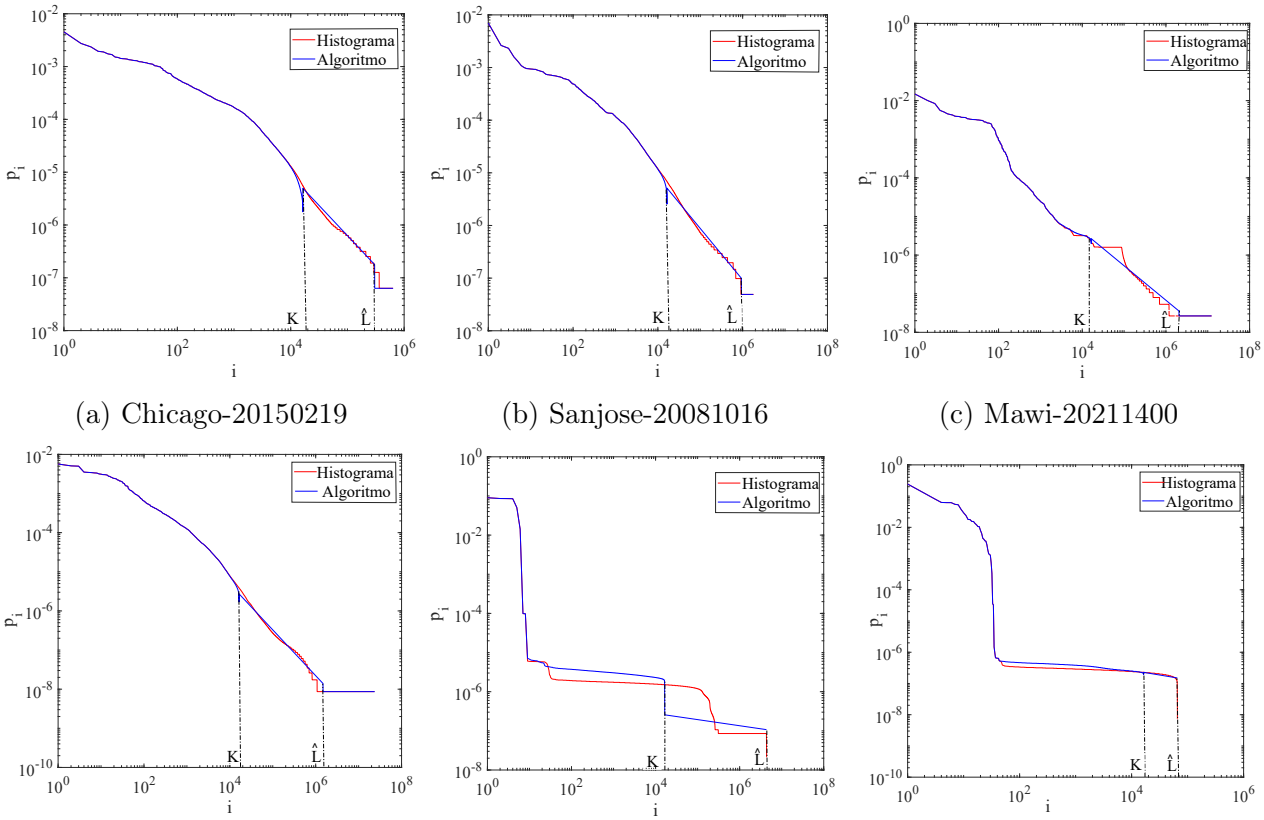
Tabla 4.5: Errores de estimación absolutos y relativos obtenidos mediante la Ecuación (4.10) y el Algoritmo 1, usando la entropía exacta como referencia.

Trazas	E_{abs_PL}	E_{abs_alg}	E_{r_PL} (%)	E_{r_alg} (%)
Chicago-20150219	0.0027	0.0000	0.3642	0.0011
Chicago-20160121	0.0090	0.0096	1.2250	1.2965
Chicago-20110608	0.0028	0.0026	0.3856	0.3480
Sanjose-20081016	0.0070	0.0035	0.8850	0.4478
Chicago-20080515	0.0109	0.0104	1.3723	1.2999
Chicago-20080319	0.0123	0.0120	1.4664	1.4210
Mawi-20211400	0.0069	0.0065	0.9261	0.8697
Mawi-20201400	0.0019	0.0016	0.2730	0.2305
Mawi-20191400	0.0034	0.0023	0.4367	0.2903
Mawi-20181400	0.0045	0.0046	0.6450	0.6579
Mawi-20171400	0.0022	0.0024	0.3218	0.3596
Mawi-20161400	0.0051	0.0073	0.7181	1.0324
Capture-7-1	0.0000	0.0027	0.0021	0.2670
Capture-17-1	0.0009	0.0001	0.0898	0.0029
Capture-35-1	0.0187	0.0159	2.8436	2.4218
Capture-60-1	0.0000	0.0001	0.0032	0.0387
Promedio	**0.0055**	**0.0051**	**0.7474**	**0.6866**

es suficientemente precisa para producir una buena estimación de la entropía.

Comparación con trabajos relacionados

Los resultados obtenidos se comparan favorablemente con los de otros algoritmos para la estimación de entropía del tráfico de red, analizados en la secciones 2.3.1 y 2.4.1. El *sketch* presentado por Lai *et al.* [53] utiliza 2 MiB de memoria y fue evaluado en dos conjuntos de 10 trazas de MawiLab de los años 2007 y 2015, logrando errores medios de estimación de entropía de 8.5 % y 11.85 %, respectivamente. En comparación, las estructuras de datos utilizadas por el algoritmo propuesto requieren 2.8 MiB de memoria, pero el error medio de estimación es más de un orden de magnitud menor y fue probado en trazas de mayor tamaño. En su trabajo, Ding *et al.* [35] utilizan un CS de 40 KiB para estimar las frecuencias de paquetes, alcanzando un error medio de estimación de entropía del 3 % en trazas con menos de 8,000 flujos y hasta 460,000 paquetes. El algoritmo propuesto utiliza un *sketch* considerablemente más grande, pero su error medio es más de 4 veces menor. Además, los resultados se obtuvieron a partir de trazas que



(a) Chicago-20150219 (b) Sanjose-20081016 (c) Mawi-20211400

(d) Mawi-20171400 (e) Capture-35-1 (f) Capture-60-1

Figura 4.9: Histogramas normalizados de probabilidades de los flujos, en escala log-log, para seis trazas. Cada gráfico compara los datos reales de la traza con las estimaciones producidas por el algoritmo propuesto. Los puntos entre 1 y K corresponden a frecuencias estimadas por el Count Sketch y capturadas por el PQA. Para valores entre K y $\hat{L}$, el algoritmo estima las frecuencias mediante un ajuste de ley de potencias, mientras que para valores superiores a $\hat{L}$, las frecuencias se aproximan como un paquete por flujo.

contienen hasta 4 órdenes de magnitud más flujos y 3 órdenes de magnitud más paquetes, lo que exige un *sketch* con dimensiones mayores.

Enfoques como *UnivMon* [61], *ElasticSketch* [62] y *LightGuardian* [63] utilizan *sketches* para rastrear las frecuencias de los flujos y otras propiedades con el fin de calcular diferentes métricas de tráfico. *UnivMon* logra un error de estimación de aproximadamente 2 % usando un *sketch* de 1 MiB. *ElasticSketch* y *LightGuardian* logran errores pequeños, inferiores al 1 %, utilizando *sketches* de entre 1 y 3 MiB. Sin embargo, han sido validados con conjuntos de datos pequeños, de menos de un millón de flujos y menos de seis millones de paquetes. El algoritmo propuesto está diseñado para operar con conjuntos de datos de decenas de millones de flujos y cientos de millones de paquetes. Al igual que *ElasticSketch*, el objetivo es estimar con precisión las frecuencias de

Tabla 4.6: Parámetros utilizados en la implementación actual de algoritmos publicados previamente y del algoritmo propuesto.

Trazas	K	p (*HLL*)	$w \times d$ (CS)	$R \times S$ (PQA)
Top-K [31]	$65,536$	-	$65,536 \times 11$	$8,192 \times 8$
Top-K_U [34]	$16,384$	11	$65,536 \times 11$	$2,048 \times 8$
Este trabajo	$16,384$	13	$65,536 \times 11$	$2,048 \times 8$

los paquetes de los top-K flujos. El PQA utilizada puede considerarse una generalización de la tabla *hash* utilizada por *ElasticSketch*. En lugar de un esquema de votación, se usan estas colas para expulsar flujos del PQA. Además, a diferencia de *ElasticSketch*, que usa un *sketch* de tipo CM para medir los flujos menos frecuentes, se estiman estas frecuencias mediante un ajuste de ley de potencia, lo que requiere menos memoria y menos recursos computacionales.

A continuación, se comparan los resultados del algoritmo con dos métodos de estimación propuestos previamente, ejecutados sobre las mismas trazas: el primer método calcula la entropía usando únicamente los top-K flujos [31], el segundo asume que las frecuencias de los flujos menos frecuentes son todas iguales [34]. Los tres algoritmos emplean un CS y un PQA para almacenar las frecuencias de los flujos top-K. El primer método, denominado `Top-`K [31], no calcula la cardinalidad de flujos. En cambio, tanto el segundo método, `Top-`K`_U` [34], como el algoritmo propuesto en este trabajo, utilizan un *sketch* HLL para estimarla. Los métodos anteriores fueron originalmente evaluados usando solo la dirección IP de origen como identificador de flujo. Por lo tanto, en esta evaluación se ajustaron sus parámetros para operar con la 5-tupla (dirección y puerto de origen, dirección y puerto de destino, y protocolo) para identificar los flujos.

En la Tabla 4.6 se resumen los parámetros de las estructuras de datos utilizadas en los tres algoritmos. Dado que `Top-`K ignora los flujos menos frecuentes, utiliza un valor de K cuatro veces mayor que los otros algoritmos para lograr un error de estimación razonable. Los tres algoritmos emplean un CS del mismo tamaño, ya que todos procesan la misma cantidad de paquetes y flujos. Finalmente, el parámetro de precisión del HLL usado en `Top-`K`_U` es ligeramente mayor que en este trabajo, porque la estimación de la distribución de ley de potencias es más sensible a la cardinalidad de flujos que la suposición comparativamente simple de frecuencias uniformes usada por `Top-`K`_U`.

En la Tabla 4.7 se resumen los errores de estimación de los tres algoritmos, relativos a la entropía exacta de los flujos, etiquetados como $E_{r\_K\_alg}$, $E_{r\_U\_alg}$ y $E_{r\_PL\_alg}$, correspondientes a `Top-`K, `Top-`K`_U` y a este trabajo, respectivamente. Los errores producidos por `Top-`K son

Tabla 4.7: Errores relativos de estimación de entropía obtenidos por algoritmos publicados previamente y el algoritmo propuesto, utilizando como referencia la entropía exacta y normalizada de los flujos.

Trazas	$E_{r\_K\_alg}$ (%) [31]	$E_{r\_U\_alg}$ (%) [34]	$E_{r\_PL\_alg}$ (%)
Chicago-20150219	11.9594	1.7941	0.0011
Chicago-20160121	15.2976	2.7291	1.2965
Chicago-20110608	15.7791	2.6868	0.3480
Sanjose-20081016	11.8173	3.4707	0.4478
Chicago-20080515	11.6513	3.0523	1.2999
Chicago-20080319	7.8095	1.3034	1.4210
Mawi-20211400	7.8125	3.6580	0.8697
Mawi-20201400	4.2731	1.0032	0.2305
Mawi-20191400	10.3220	0.8585	0.2903
Mawi-20181400	4.0007	1.3604	0.6579
Mawi-20171400	21.0304	3.1580	0.3596
Mawi-20161400	8.6938	2.4907	1.0324
Capture-7-1	4.2800	0.2102	0.2670
Capture-17-1	0.1113	0.0137	0.0029
Capture-35-1	38.6209	2.4521	2.4218
Capture-60-1	0.5659	0.1286	0.0387
Promedio	**10.8763**	**1.8981**	**0.6866**

significativamente mayores que los de los otros dos métodos, con un máximo de 38,6 % y un promedio de 10,9 %. El error promedio de estimación de `Top-`K`_U` es de 1,9 %, pero supera el 3 % en cuatro de las 16 trazas, y el 2 % en ocho de ellas. En comparación, el error del método actual tiene un promedio de 0,69 %, y sólo supera el 1,5 % en una traza y el 1 % en cinco.

Para comparar el tiempo de ejecución y el uso de memoria de los algoritmos se construyeron implementaciones en C++ y se compilaron usando GCC versión 9.4.0. Se utilizó la biblioteca *PcapPlusPlus* para analizar los archivos de las trazas de paquetes. También se implementó un algoritmo que calcula la entropía exacta usando un mapa desordenado estándar para mantener un contador de paquetes por flujo. Los experimentos se ejecutaron en un procesador Intel Core i9-10980XE de 3 GHz con 128 GiB de memoria y el sistema operativo Ubuntu 20.04 LTS.

En la Tabla 4.8 se muestra la memoria requerida para calcular la entropía de los flujos utilizando el algoritmo exacto y los tres métodos de estimación. Se utilizó la función *malloc_ count*

Tabla 4.8: Uso máximo de memoria del algoritmos que calcula la entropía exacta de los flujos, métodos de estimación anteriores y el método propuesto en este trabajo.

Trazas	Memoria (MiB)			
	Exacta	`Top-`K	`Top-`K`_U`	Este trabajo
Chicago-20150219	68.50	4.31	3.14	3.15
Chicago-20160121	97.00	4.31	3.14	3.15
Chicago-20110608	178.39	4.31	3.14	3.15
Sanjose-20081016	201.36	4.31	3.14	3.15
Chicago-20080515	104.06	4.31	3.14	3.15
Chicago-20080319	44.62	4.31	3.14	3.15
Mawi-20211400	1,412.74	4.31	3.14	3.15
Mawi-20201400	4,809.08	4.31	3.14	3.15
Mawi-20191400	3,549.51	4.31	3.14	3.15
Mawi-20181400	3,049.20	4.31	3.14	3.15
Mawi-20171400	2,498.51	4.31	3.14	3.15
Mawi-20161400	3,034.27	4.31	3.14	3.15
Capture-7-1	1,221.23	4.31	3.14	3.15
Capture-17-1	5,754.21	4.31	3.14	3.15
Capture-35-1	479.80	4.31	3.14	3.15
Capture-60-1	7.18	4.31	3.14	3.15

para calcular el valor máximo de memoria dinámica usada para implementar los *sketches*: CS y HLL, el PQA, y el mapa. Los tres algoritmos de estimación utilizan una cantidad constante de memoria, mientras que la memoria usada por el algoritmo exacto es hasta tres órdenes de magnitud mayor, proporcional a la cardinalidad de la traza. `Top-`K utiliza más memoria que `Top-`K`_U` y el método propuesto, debido a que usa un PQA más grande. La memoria requerida por el algoritmo propuesto es solo 10 KiB mayor que la de `Top-`K`_U`, ya que, aunque usa un *sketch* HLL más grande, su uso de memoria está dominado por el CS y el PQA.

El tiempo de cómputo del algoritmo puede dividirse en dos partes: el tiempo para procesar cada paquete entrante, que determina su *throughput* (rendimiento), y el tiempo para estimar la entropía a partir del conjunto de contadores de frecuencia, que determina su latencia. Los tres algoritmos de estimación procesan los paquetes de la misma manera, por lo tanto, su *throughput* es el mismo. En la evaluación realizada se procesa un paquete en aproximadamente 650 ns para los tres algoritmos, medido usando la biblioteca estándar *chrono* de C++. En la Tabla 4.9 se muestra la latencia de estimación de los algoritmos. El algoritmo exacto es el más lento porque

Tabla 4.9: Latencia de las implementaciones en C++ de los algoritmos de estimación de entropía propuestos previamente y el método presentado en este trabajo.

Trazas	Tiempo de estimación de entropía (ms)			
	Exacta	Top-K[31]	Top-K_U[34]	Este trabajo
Chicago-20150219	122.07	2.22	0.68	2.44
Chicago-20160121	174.54	2.21	0.68	2.48
Chicago-20110608	346.07	2.23	0.67	2.42
Sanjose-20081016	376.38	2.21	0.67	2.43
Chicago-20080515	187.48	2.23	0.67	2.42
Chicago-20080319	79.50	2.22	0.67	2.30
Mawi-20211400	3,319.74	2.20	0.67	2.30
Mawi-20201400	11,621.67	2.11	0.65	2.48
Mawi-20191400	9,142.33	2.19	0.67	2.36
Mawi-20181400	7,455.51	2.18	0.67	2.40
Mawi-20171400	5,737.66	2.14	0.65	2.48
Mawi-20161400	7,257.19	2.17	0.67	2.47
Capture-7-1	2,879.36	2.22	0.66	1.94
Capture-17-1	14,535.02	2.11	0.65	1.99
Capture-35-1	956.28	2.23	0.66	2.17
Capture-60-1	6.16	2.13	0.68	2.12

utiliza las frecuencias de todos los flujos, mientras que los tres algoritmos de estimación solo usan los top-K flujos. Aunque computacionalmente más simple, la latencia de Top-K es mayor que la de Top-K_U porque usa un valor de K más grande. La latencia de Top-K_U es menor que la del método propuesto porque emplea una expresión más simple para calcular la entropía. La latencia del algoritmo es similar a la de Top-K, pero su error medio de estimación es 15.9 veces menor. Como se mostrará a continuación, el acelerador hardware puede reducir eficientemente el tiempo para calcular la estimación de entropía [112, 113].

4.6.3. Rendimiento del acelerador

El acelerador hardware se implementó en el plano de datos utilizando una tarjeta Xilinx Alveo U280, que contiene un FPGA Virtex UltraScale+ XCU280. Esta tarjeta es compatible con NetFPGA PLUS, un *framework* de código libre para dispositivos de red inteligentes. El

acelerador se diseñó en SystemVerilog a nivel de transferencia de registros (*Register-Transfer Level*, RTL), y se implementó con la herramienta de síntesis Xilinx Vivado, versión 2022.2.

Como se explicó en la Sección 4.5, el acelerador utiliza solo 512 de las 2048 colas del PQA para calcular el valor de $\hat{\alpha}$. Además, los 8 bloques de memoria que componen el PQA se ordenan de forma independiente y en paralelo, en lugar de como un solo bloque. La combiandción de estas dos optimizaciones reducen el tiempo de cálculo de $\hat{\alpha}$ en un factor de 32, pero introducen un error en su estimación. Además, dado que el soporte de hardware aritmético en FPGAs está limitado a operaciones de suma y multiplicación enteras, el acelerador utiliza una representación numérica de punto fijo de 60 bits y aproximaciones basadas en tablas (LUTs) para funciones como logaritmos y exponenciación. En la Tabla 4.10 se comparan los errores de estimación de entropía absolutos y relativos del algoritmo y del acelerador, usando la entropía exacta como referencia. Como se muestra en la tabla, las aproximaciones introducidas por el acelerador aumentan su error de estimación de 0.0011 % a 1.097 % en el peor de los casos, en comparación con el algoritmo. Sin embargo, el error medio de estimación para las 16 trazas aumenta solo en 0.07 %, pasando de 0.69 % a 0.76 %. La mayor parte del error se debe a las dos optimizaciones aplicadas a el PQA.

La arquitectura del módulo de procesamiento de paquetes se diseñó para maximizar la frecuencia de reloj, y así maximizar la velocidad de línea a la que puede operar el acelerador. En la implementación propuesta, este módulo funciona a 400 MHz, lo que permite alcanzar una velocidad de línea superior a 204 Gbps en el peor caso de paquetes de 64 bytes. La arquitectura del módulo de estimación fue diseñada para optimizar la latencia y el uso de recursos, y opera a 200 MHz. Produce un valor de entropía con una latencia de 16 μs, dominada por el bloque de ordenamiento *radix* (12 μs) y la estimación por mínimos cuadrados de $\hat{\alpha}$ (3 μs).

En la Tabla 4.11 se muestra la utilización de recursos del FPGA para los bloques principales del acelerador. Solo se utiliza un 1.3 % de los flip-flops disponibles. La utilización de recursos lógicos (LUT) y aritméticos (DSP) es de aproximadamente 3.3 %, la mayoría utilizados en el módulo que calcula $\hat{H}_2$. El uso de memoria en el chip es más elevado: 13.2 % de la BRAM y 9.2 % de la URAM. Esta memoria se usa principalmente para implementar: el *sketch* de cuentas, la estructura del PQA, los buffers usados por el *radix-sort* y las tablas de aproximación para el logaritmo. A pesar de esto, la utilización general de recursos es baja, lo que contribuye a reducir los retardos de enrutamiento y a lograr una mayor frecuencia de reloj en dispositivos reconfigurables. Además, el acelerador deja la mayor parte del hardware del FPGA libre, permitiendo así implementar otras tareas de medición y control de red en paralelo.

Tabla 4.10: Errores de estimación absolutos y relativos obtenidos por el Algoritmo 1 y el acelerador, utilizando la entropía exacta como referencia.

Trazas	E_{abs_alg}	E_{abs_acc}	E_{r_alg} (%)	E_{r_acc} (%)
Chicago-20150219	0.0000	0.0083	0.0011	1.0973
Chicago-20160121	0.0096	0.0031	1.2965	0.4116
Chicago-20110608	0.0026	0.0067	0.3480	0.9035
Sanjose-20081016	0.0035	0.0043	0.4478	0.5457
Chicago-20080515	0.0104	0.0029	1.2999	0.3591
Chicago-20080319	0.0120	0.0037	1.4210	0.4360
Mawi-20211400	0.0065	0.0005	0.8697	0.0685
Mawi-20201400	0.0016	0.0063	0.2305	0.8806
Mawi-20191400	0.0023	0.0062	0.2903	0.7915
Mawi-20181400	0.0046	0.0091	0.6579	1.3133
Mawi-20171400	0.0024	0.0009	0.3596	0.1460
Mawi-20161400	0.0073	0.0136	1.0324	1.9182
Capture-7-1	0.0027	0.0017	0.2670	0.1684
Capture-17-1	0.0001	0.0066	0.0029	0.6656
Capture-35-1	0.0159	0.0003	2.4218	2.4628
Capture-60-1	0.0001	0.0003	0.0387	0.0120
Promedio	**0.0051**	**0.0047**	**0.6866**	**0.7612**

Tabla 4.11: Utilización de recursos del FPGA por núcleo (kernel).

Recursos	LUT	Flip-flop	BRAM	URAM	DSP
HLL	652	630	1.5	0	18
CS	3,937	3,266	0	88	12
PQA	545	1,185	32	0	0
$\hat{H}_1$	2,216	2,579	80.5	0	26
$\hat{H}_2$	30,691	22,497	171.5	0	170
$\hat{H}_3$	65	40	0	0	3
Total	**38,241**	**30,335**	**285.5**	**88**	**229**
Utilización (%)	**3.2**	**1.3**	**13.2**	**9.2**	**3.3**

El consumo de potencia del acelerador implementado en el FPGA Alveo U280 está compuesto por una potencia estática de 3.303W y una potencia dinámica de 7.231W, lo que da como resultado un consumo total aproximado de 10.534W. La potencia estática corresponde al

consumo base del dispositivo, asociado principalmente a las corrientes de fuga inherentes a la tecnología de fabricación, y se mantiene constante independientemente de la actividad del acelerador. Por su parte, la potencia dinámica refleja el costo energético de la ejecución del diseño, ya que depende de la conmutación de los recursos lógicos, memorias y bloques de cómputo utilizados, así como de la frecuencia de operación y la carga de trabajo procesada. En este contexto, la potencia total obtenida representa alrededor de un 5 % del presupuesto máximo de potencia del FPGA U280, cuya potencia máxima de diseño térmico (*Thermal Design Power*, TDP) puede superar los 225W, lo que evidencia que el acelerador propuesto logra un uso eficiente de los recursos y del consumo energético.

4.7. Discusión

En este capítulo se presentó un método y un algoritmo de *streaming* para la estimación de la entropía empírica de Shannon en intervalos discretos, junto con su correspondiente acelerador implementado en FPGA. Los resultados obtenidos en trazas de red reales mostraron que el algoritmo logra errores relativos promedio inferiores al 1 %, con un consumo de memoria hasta tres órdenes de magnitud menor que los métodos exactos. Además, la arquitectura propuesta alcanzó velocidades de línea superiores a 200 Gbps y una latencia de estimación del orden de microsegundos, demostrando la viabilidad de aplicar *sketches* y colas de prioridad en entornos de tráfico masivo. Estos resultados permiten generalizar que los algoritmos basados en *sketches* no solo reducen significativamente los requerimientos de memoria, sino que también son altamente paralelizables y adaptables a arquitecturas hardware, lo que los hace adecuados para escenarios de monitoreo en tiempo real en redes de alta velocidad. No obstante, una limitación importante de este enfoque es que la estimación se restringe a intervalos fijos, lo que impide capturar adecuadamente la naturaleza dinámica del tráfico, donde los eventos recientes tienen mayor relevancia. En consecuencia, esta restricción motiva la extensión hacia el modelo de ventanas deslizantes, presentado en el Capítulo 5, el cual busca mantener actualizadas las estimaciones de entropía en intervalos traslapados, conciliando precisión, eficiencia en el uso de memoria y capacidad de operar a velocidades de línea.

Capítulo 5. Entropía en ventanas de tiempo deslizantes

5.1. Introducción

En el presente capítulo se adapta el algoritmo de estimación de entropía empírica de Shannon de flujos de red, propuesto en el capítulo anterior, para su uso en ventanas de tiempo deslizantes. Específicamente, se estima la entropía en cada ventana de tiempo de tamaño W cada s unidades de tiempo, reflejando la distribución de los flujos de red observados en los últimos W instantes. El algoritmo adaptado utiliza *sketches* para estimar la frecuencia y la cardinalidad en cada ventana; y arreglos de colas de prioridad para detectar y almacenar los flujos top-K, aquellos con las frecuencias más altas. Además, se propone y evalua el rendimiento de una arquitectura en FPGA para su aceleración hardware. El algoritmo se evalúa en un conjunto de trazas reales disponibles en el repositorio público de MawiLab y se describe el rendimiento del acelerador.

5.2. Modelo de ventanas deslizantes

Un flujo de datos es una secuencia de paquetes con atributos comunes que llega en tiempo real, como $(\langle e_1 t_1 \rangle, \langle e_2 t_2 \rangle, ..., \langle e_i t_i \rangle)$. La marca de tiempo t_i representa el momento en que el elemento e_i llega al dispositivo de red. Cada elemento e_i corresponde a un atributo específico o identificador de flujo. En este capítulo se utiliza la 5-tuple para identificar los paquetes que pertenecen al mismo flujo: direcciones IP de origen y destino, protocolos de la capa de transporte y puertos de origen y destino.

Un enfoque común para la medición de flujos de datos consiste en dividir el flujo en ventanas y aplicar los algoritmos de medición en cada una de ellas. En [114], los autores definieron varios mecanismos según el tamaño y los límites de la ventana, tales como basados en el tiempo, en el conteo de elementos y en puntos de referencia (*landmark-based*). En el mecanismo basado en tiempo, se agrupan los flujos de datos que llegan durante un intervalo de tiempo fijo y en el basado en elementos se agrupa un número fijo de flujos, sin importar cuándo lleguen. Este último mecanismo es menos intuitivo si se necesita correlacionar eventos con el tiempo real y el tiempo cubierto por cada ventana puede variar mucho si la tasa de llegada de datos es

variable. En el mecanismo basado en puntos de referencia todas las observaciones se agrupan desde un punto fijo de referencia temporal, hasta el instante actual. Es particularmente útil en aplicaciones donde se desea medir el comportamiento acumulado desde un evento significativo.

En [114], los autores también identifican los tipos de ventanas encontrados en la literatura para aplicar estos mecanismos, como las ventanas por lotes (*tumbling*), atenuadas (*damped*) y deslizantes. Estas ventanas se basan en la suposición de que los datos más recientes son los más importantes, lo que permite limitar la cantidad de datos a procesar y mantener mediciones de red constantemente actualizadas. Las ventanas por lotes dividen el flujo continuo en intervalos consecutivos, disjuntos y de tamaño fijo, como se analizó en el capítulo anterior. Cada ventana cubre una porción de tiempo o una cantidad específica de elementos, y una vez finalizada, la ventana se cierra y comienza una nueva, sin superposición entre ellas. Las ventanas atenuadas, también conocidas como ventanas con decaimiento exponencial, son un modelo donde todos los datos históricos se consideran, pero su influencia disminuye progresivamente con el tiempo. En lugar de eliminar explícitamente los datos antiguos, cada elemento tiene un peso que decrece con el tiempo, generalmente de forma exponencial. El solapamiento de los datos no es total, sino parcial. Es útil cuando se desea una respuesta suave pero adaptable, preservando información pasada con menor importancia. Las ventanas deslizantes son una técnica de segmentación de flujos de datos en la que se mantiene una ventana de tamaño fijo (por tiempo o por número de elementos) que se actualiza periódicamente al avanzar el flujo. En cada paso o deslizamiento, se descartan los elementos más antiguos y se incorporan los nuevos. Este tipo de ventana permite capturar el comportamiento reciente y dinámico de los datos y es ampliamente usado en monitoreo de redes. En este capítulo se estima la entropía utilizando ventanas deslizantes de tamaño fijo W que se deslizan $s = \frac{W}{10}$ unidades de tiempo, como se muestra en la Figura 5.1.

5.3. Método

En la presente sección y la siguiente adaptamos el método y algoritmo propuesto en las Secciones 4.3 y 4.4 respectivamente, para la estimación de entropía empírica en ventanas de tiempo deslizante. El método anterior se basa en dos propiedades observadas en las trazas de tráfico de red utilizadas. Primero, una pequeña fracción de los flujos concentra la mayoría de las frecuencias altas, contribuyendo significativamente más al valor de la entropía empírica que los flujos menos frecuentes. Segundo, la distribución de una fracción significativa de los elementos sigue aproximadamente una ley de potencias. El algoritmo que implementa el método utiliza un *sketch* HLL para estimar la cardinalidad de los flujos, un CS para la estimación de frecuencias

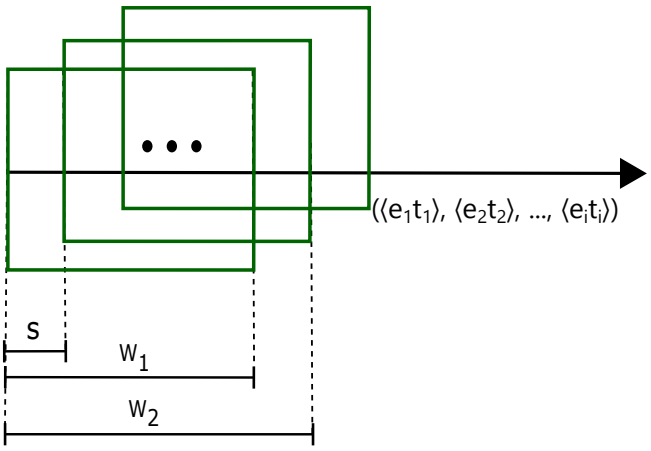


Figura 5.1: Modelo de ventana deslizante utilizado en la estimación de entropía empírica.

y un PQA que detecta y almacena los top-K flujos más frecuentes.

Las modificaciones propuestas se basan en observaciones empíricas realizadas en ventanas de tiempo de tamaño W y desplazamiento $s = \frac{W}{10}$; donde la cardinalidad en cada intervalo s es en promedio un $\frac{1}{9}$ de la cardinalidad de toda la ventana. Por lo tanto, aproximadamente el 90 % de los flujos se traslapan. Esta observación permite estimar las frecuencias de la ventana W usando la unión de diez *sketches* de cuentas de menor dimensión, uno en cada intervalo s. La implementación en hardware requiere un *sketch* adicional para evitar la pérdida de datos durante la operación de borrado. Por lo tanto, se utilizan once *sketches* de manera cíclica, borrando el más antiguo en cada deslizamiento después de la primera ventana, hasta procesar todos los paquetes, como se muestra en la Figura 5.2. La inserción en cada intervalo s se produce en solo un *sketch* a la vez. La unión de los *sketches* reduce el uso de la memoria a la mitad respecto al uso de un *sketch* en cada ventana, como se muestra en la Sección 5.6.1.

La estimación de cardinalidad se obtiene de forma similar, usando once *sketches* HLL de forma cíclica, pero uno en cada ventana W. Dado que la operación de combinación en este caso se realiza entre *sketches* de igual dimensión seleccionando el máximo entre *buckets*, no hay reducción en sus dimensiones. En la Figura 5.3, se muestra como en la ventana W_2 los identificadores de flujo se insertan en varios *sketches* a la vez del HLL_2 al HLL_{11}, porque el HLL_1 está siendo borrado. La estimación se realiza usando el más antiguo en que se insertaron datos, el HLL_2.

Para la detección y almacenamiento de flujos top-K y el cálculo del número de paquetes en cada ventana, también se utilizan once PQA y once contadores respectivamente, de forma cíclica y similar a la estimación de cardinalidad. En el caso de los PQA, su uso en intervalos s

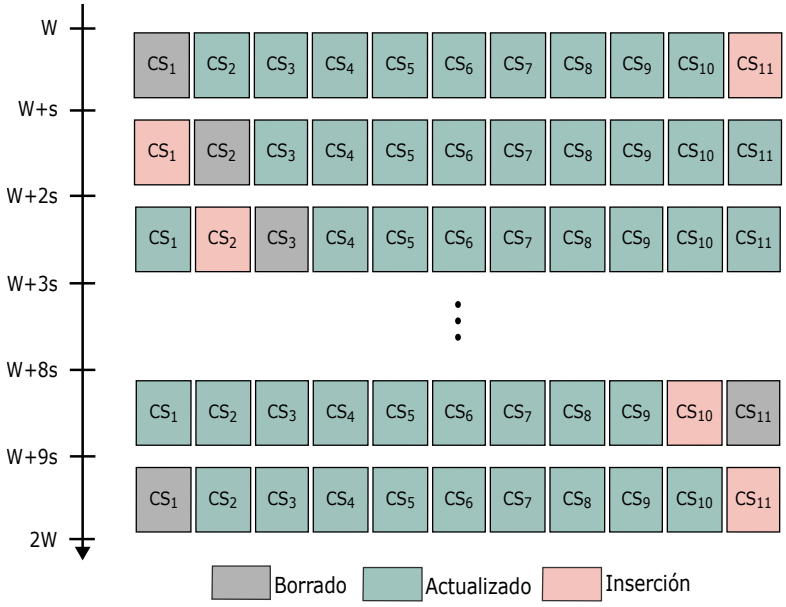


Figura 5.2: Esquema en bloques de la estructura cíclica formada por once *sketches* CS utilizada para la estimación de frecuencia en ventanas de tiempo deslizantes.

puede detectar top-K que pueden no serlo en la ventana, afectando la precisión del algoritmo, por lo que utilizamos 11 PQA, una en cada ventana.

5.4. Algoritmo de estimación de entropía en ventanas de tiempo deslizantes

En el Algoritmo 2 se muestra el procesamiento en *streaming* de los paquetes de red en ventanas de tiempo de tamaño W y deslizamiento $s = \frac{W}{10}$. En las líneas 2-5 se inicializan en cero las estructuras de datos: *CardSketch*, *FreqSketch* y *PQ*, cada una formada por 11 subestructuras, para estimar la cardinalidad de los flujos $\hat{N}$, sus frecuencia y almacenar los K flujos más frecuentes, respectivamente. Además, se inicializa M en cero, que utiliza 11 contadores para determinar el número de paquetes en cada ventana.

En la línea 7 se obtiene el tiempo de llegada de cada paquete $t(e)$. Seguido, se verifica el tiempo transcurrido para actualizar las estructuras de datos. La inserción del identificador de flujo de cada paquete e en *CardSketch* se realiza primero en un *sketch*, pasado s unidades de tiempo los elementos se insertan en el primer y en el segundo, seguido en el primero, segundo y tercero y así sucesivamente, como se muestra en la línea 17. El contador de paquetes se

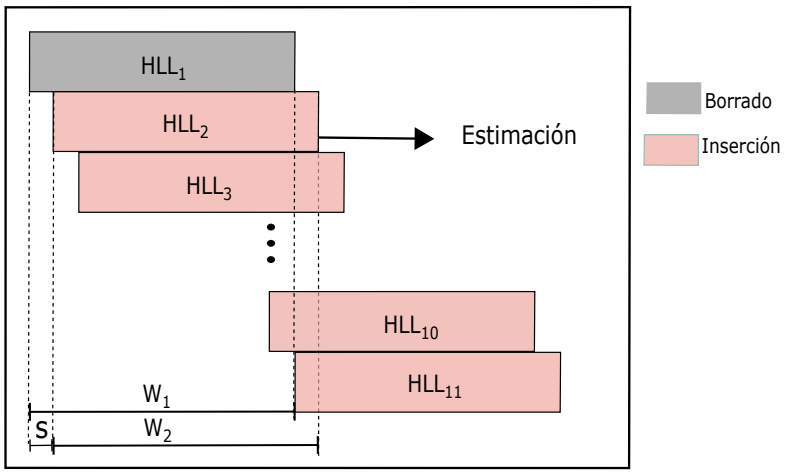


Figura 5.3: Esquema en bloques de la estructura cíclica formada por once HLL utilizada para la estimación de cardinalidad en ventanas de tiempo deslizantes.

incrementa de forma similar. Sin embargo, para la estructura *FreqSketch* en un intervalo de tiempo s el identificador de flujo e solo se inserta en un *sketch*. La actualización de *FreqSketch* se muestra en la línea 20. El algoritmo utiliza variables como: $cont\_slide$ para determinar el número de desplazamientos de tamaño s en una ventana; idx_start para determinar que estructura o contador se borra y idx_insert_freq como índice para indicar el siguiente *sketch* de cuentas donde deben insertarse elementos. En la línea 21, verifica si ha pasado el tiempo de una ventana W para estimar la cardinalidad y frecuencia de los flujos en las líneas 22-30. En la estimación de cardinalidad se considera la estimación de un *sketch* por ventana y en la de frecuencia la unión de diez *sketches* de igual dimensión. En la línea 31, la estimación de frecuencia de cada paquete y el identificador de flujo e se insertan en la PQ de la ventana actual. En las líneas 32-38 se borran los *sketches*, la PQ y el contador más antiguo y se van rotando de manera circular.

Después del procesamiento de los paquetes se estima la entropía utilizando $\hat{N}$, M y el contenido de la PQ de cada ventana, como se muestra en el Algoritmo 3. En las líneas 2-4, se lee el contenido de la PQ para calcular K, el número de elementos frecuentes almacenados en el arreglo de colas y se computa $\hat{H}_1$, utilizando la Ecuación (4.6).

En la línea 6, el algoritmo calcula $\hat{\alpha}$ una estimación del parámetro exponencial α de la distribución de ley de potencias. Esta distribución puede representarse como un ajuste lineal en escala log-log del histograma de frecuencias, y $\hat{\alpha}$ corresponde a la pendiente del ajuste lineal calculado mediante el método de mínimos cuadrados [99] y la Ecuación (4.11), como se explicó en la sección 4.4. El método de mínimos cuadrados utiliza las frecuencias almacenadas en la

estructura PQ en orden descendente.

En la línea 7 se estima $\hat{C}$ usando también el método de mínimos cuadrados como:

$$\hat{C} = \frac{\sum_{i=1}^{K}(\log \hat{m}_i) - \hat{\alpha}\sum_{i=1}^{K}\log i}{\log K}. \tag{5.1}$$

El valor de $\hat{C}$ permite estimar posteriormente $\hat{L}$, el valor mínimo estimado de i para el cual $\hat{m}_i = 1$ en el histograma. Utilizando la ecuación de la recta en la escala log-log, donde $\hat{\alpha}$ es la pendiente y $\hat{C}$ es la intersección con el eje de las frecuencias, $\hat{L}$ se estima en la línea 8 como:

$$\hat{L} = 2^{-\frac{\hat{C}}{\hat{\alpha}}}. \tag{5.2}$$

Posteriormente, el algoritmo recalcula el valor de $\hat{C}$ considerando que la suma de todas las probabilidades debe ser igual a uno. Esta variación con respecto al Algoritmo 1, donde $\hat{C}$ se calculaba solo una vez, considerando que la suma de las probabilidades debe ser uno y $\hat{L}$ con una ecuación lineal simple basada en los puntos $(K, \frac{\hat{m}_K}{M})$ y $(\hat{L}, \frac{1}{M})$, permite obtener mejores estimaciones de $\hat{L}$ y por tanto de la entropía normalizada.

Generalmente, $L < N$ y el algoritmo producirá estimaciones tales que $\hat{L} < \hat{N}$. Para este caso frecuente, en la línea 10, el algoritmo recalcula los valores de $\hat{C}$, considerando que la suma de todas las probabilidades deben ser igual a uno, como se mostró en la Ecuación (4.13). Se usó esta expresión para normalizar la suma de las frecuencias en la estimación y el formalismo continuo de la ley de potencias, descrito en la Sección 4.4, para calcular el valor de $\hat{C}$ mediante la Ecuación (4.16). El formalismo continuo de la ley de potencias simplifica los cálculos discretos. En particular, el término $\sum_{i=K+1}^{\hat{L}} i^{-\hat{\alpha}}$ que es computacionalmente costoso, ya que requiere una exponenciación para cada uno de sus $\hat{L} - \hat{K}$ términos, y normalmente $\hat{L} - \hat{K} \gg \hat{K}$. A continuación, en la línea 11, el algoritmo calcula el valor de $\hat{H}_2$ usando la Ecuación (4.20). En la línea 12 usando la Ecuación (4.9) el algoritmo calcula $\hat{H}_3$ y en la línea 13 $\hat{H}$ como $\hat{H} = \hat{H}_1 + \hat{H}_2 + \hat{H}_3$.

Las líneas 14-17 del algoritmo tratan el caso en el que $\hat{L} \geq \hat{N}$. En este caso, se ignoran las frecuencias estimadas de los flujos más allá de $\hat{N}$, es decir, se usa $\hat{L} = \hat{N}$. Así, $\hat{H}_3 = 0$ y el cálculo de $\hat{C}$ se reduce. En la línea 15, el algoritmo calcula $\hat{C}$ usando la Ecuación (4.22). En la línea 16 se estima $\hat{H}_2$ usando la Ecuación (4.24). La línea 17 calcula $\hat{H} = \hat{H}_1 + \hat{H}_2$ cuando $\hat{L} \geq \hat{N}$. Finalmente, el algoritmo calcula $\hat{H}_{norm}$ de la ventana en la línea 18 usando la Ecuación (4.10).

Algoritmo 2: Procesamiento de los paquetes de red en ventanas de tiempo deslizantes.

Input: e, $t(e)$, tamaño de la cola de prioridad K_{pq}, dimensiones del *sketch* de frecuencias d y w, precisión del *sketch* de cardinalidad p, función hash h, tamaño de la ventana W, deslizamiento s

```
// Inicialización de las estructuras de datos
```

$K \leftarrow 0$, $t \leftarrow 0$, $cont\_slide \leftarrow 0$ $idx\_start \leftarrow 1$ $idx\_insert\_freq \leftarrow 1$, $M_{actual} \leftarrow 0$
FreqSketch$[1, .., 11]$.*inicialización* (d, w, h)
CardSketch$[1, .., 11]$.*inicialización* $(p; h)$
PQ$[1, .., 11]$.*inicialización* (K)
M$[1, .., 11]$.*inicialización*
foreach *paquete* $e \in stream$ **do**
  $t_{actual} \leftarrow t(e)$
  if $t_{actual} \geq t + s$ **then**
    $cont\_slide \leftarrow cont\_slide + 1$
    if $cont\_slide > 10$ **then**
      $cont\_slide \leftarrow 10$
    $t \leftarrow t + s$
    for $i \leftarrow 0$ **to** $cont\_slide - 1$ **do**
      $idx \leftarrow idx\_start + i$
      if $idx > 11$ **then**
        $idx \leftarrow idx - 11$
      CardSketch$[idx]$.*actualización(e)*
      $M[idx] \leftarrow M[idx] + 1$
    $idx\_insert\_freq \leftarrow (idx\_insert\_freq \text{ mód } 11) + 1$
    FreqSketch$[idx\_insert\_freq]$.*actualización(e)*
    if $cont\_slide == 10$ **then**
      $\hat{N}$[idx_start] $\leftarrow$ *CardSketch*$[idx\_start]$.*estimación()*
      $M_{actual} \leftarrow$ M$[idx_start]$
      $indices\_freq \leftarrow [\,]$
      for $i \leftarrow 0$ **to** 9 **do**
        $idx \leftarrow idx\_insert\_freq - i$
        if $idx < 1$ **then**
          $idx \leftarrow idx + 11$
        $indices\_freq$.append(idx)
      $\hat{m}_i \leftarrow$ *FreqSketch.fusión(indices_freq)*
      PQ$[idx\_start]$.*actualización*$(e,$ *FreqSketch.fusión(*`índices_freq`*))*
      CardSketch$[idx\_start]$.*borrado()*
      FreqSketch$[idx\_start]$.*borrado()*
      PQ$[idx\_start]$.*borrado()*
      $M[idx\_start] \leftarrow 0$
      $idx\_start \leftarrow idx\_start + 1$
      if $idx\_start > 11$ **then**
        $idx\_start \leftarrow 1$

return M, $\hat{N}$, PQ

Algoritmo 3: Estimación de la entropía empírica normalizada en ventanas de tiempo deslizantes.

Input: PQ, $\hat{N}$, M, tamaño de la ventana W, deslizamiento s
Output: entropía normalizada $\hat{H}_{norm}$

foreach *ventana* W ***in*** *stream* **do**
  `// Contribución de los elementos top-`K` más frecuentes`
  foreach $\hat{m}_i$ ***in*** PQ **do**
    $K \leftarrow K + 1$
    $\hat{H}_1 \leftarrow$ Estimación usando M, m_i y Ecuación (4.6)
  `// Parámetros de la distribución de la ley de potencias`
  SortedPQ $\leftarrow$ *Sort(PQ)*
  $\hat{\alpha} \leftarrow$ Estimación usando K, $\hat{m}_i$, i y Ecuación (4.11)
  $\hat{C} \leftarrow$ Estimación usando K, $\hat{m}_i$, i, $\hat{\alpha}$ y Ecuación (5.1)
  $\hat{L} \leftarrow$ Estimación usando $\hat{\alpha}$, $\hat{C}$ y Ecuación (5.2)
  `// Estimación de entropía`
  if $\hat{L} < \hat{N}$ **then**
    $\hat{C} \leftarrow$ Estimación usando M, $\hat{N}$, K, $\hat{m}_i$, $\hat{\alpha}$, $\hat{L}$ y Ecuación (4.16)
    $\hat{H}_2 \leftarrow$ Estimación usando K, $\hat{\alpha}$, $\hat{L}$, $\hat{C}$ y Ecuación (4.20)
    $\hat{H}_3 \leftarrow$ Estimación usando M, $\hat{N}$, $\hat{L}$ y Ecuación (4.9)
    $\hat{H} \leftarrow \hat{H}_1 + \hat{H}_2 + \hat{H}_3$
  else
    $\hat{C} \leftarrow$ Estimación usando M, $\hat{N}$, K, $\hat{m}_i$, $\hat{\alpha}$ y Ecuación (4.22)
    $\hat{H}_2 \leftarrow$ Estimación usando K, $\hat{N}$, $\hat{\alpha}$, $\hat{C}$ y Ecuación (4.24)
    $\hat{H} \leftarrow \hat{H}_1 + \hat{H}_2$
  `// Estimación y retorno de la entropía normalizada de una ventana`
  $\hat{H}_{norm} \leftarrow$ Estimación usando $\hat{H}$, $\hat{N}$ y Ecuación (4.10)
  return $\hat{H}_{norm}$

5.5. Arquitectura en FPGA

En la Figura 5.4 se muestra la arquitectura del acelerador de estimación de entropía en ventanas de tiempo deslizantes, el cual está diseñado para dispositivos FPGA. Está compuesta por dos módulos principales: el `módulo de procesamiento de paquetes en ventanas deslizantes`, que implementa el Algoritmo 2 para estimar las frecuencias de los top-K, la cardinalidad de los flujos y el número de paquetes; y el `módulo de estimación de entropía` que utiliza los datos recolectados por el primer módulo.

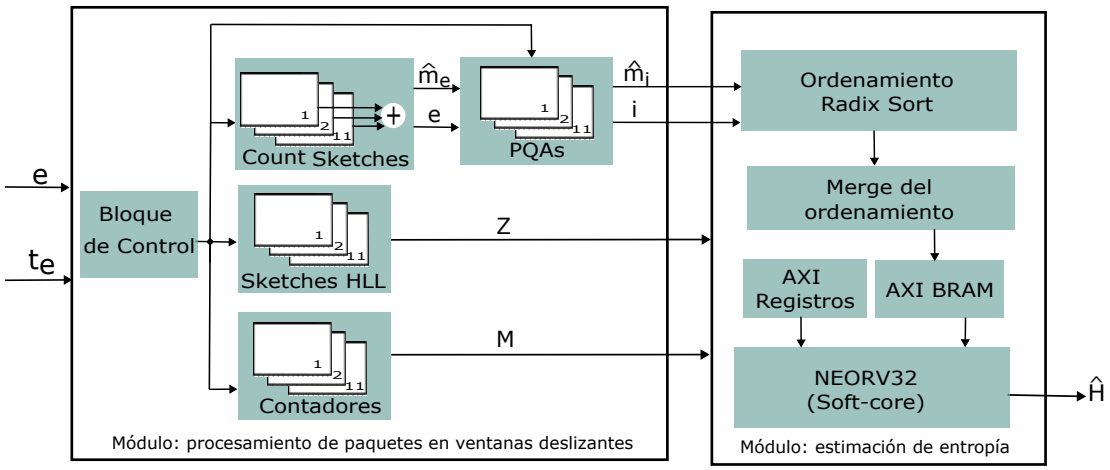


Figura 5.4: Arquitectura del acelerador hardware para la estimación de entropía en ventanas de tiempo deslizantes.

5.5.1. Módulo de procesamiento de paquetes en ventanas de tiempo deslizantes

El `módulo de procesamiento de paquetes en ventanas deslizantes` recibe el identificador de flujo e y *timestamp* o marca de tiempo t_e de cada paquete. La entrada t_e utiliza 32 bits y el identificador e 104 bits, que se construye concatenando cinco campos del encabezado del paquete: dos de 32 bits correspondientes a las direcciones IP de origen y destino, dos de 16 bits para los puertos de origen y destino y uno de 8 bits para el protocolo.

El `Bloque de Control` gestiona la inserción de un paquete válido en los *sketches* de cuentas y de cardinalidad, los contadores y los PQAs utilizados. Además, controla el borrado de los *sketches*, el contador y PQA más antiguo, una vez que pasó la primera ventana. Para ello, recibe el t_e y una señal que indica la llegada de un paquete válido. El bloque entrega dos señales de control para inserción y borrado, cada una es una palabra de 11 bits. La señal de borrado es la misma para todos y mantiene en cada desplazamiento sólo un bit activo en 1, indicando el *sketch*, contador y PQA que debe borrarse. La señal de inserción para los *sketches* de frecuencias mantiene sólo un bit activo en 1, en cada desplazamiento. Sin embargo, la señal de inserción para los *sketches* de cardinalidad, los contadores y PQAs, mantiene 10 bits activo en 1, después de la primera ventana. Las señales de inserción y borrado nunca mantienen el mismo bit activo, para evitar la pérdida de datos. De forma general, estas señales activan y desactivan diferentes estructuras según el tiempo de forma cíclica, basadas en contadores y desplazamientos *one-hot*.

El bloque `Count Sketches` estima las frecuencias en cada ventana de tamaño W; utilizando

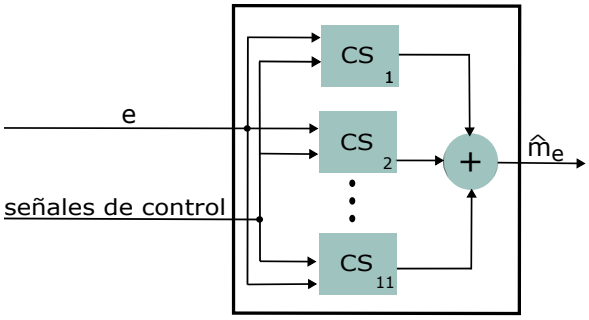


Figura 5.5: Módulo de estimación de frecuencias usando la unión de CSs.

once CS, uno en cada intervalo s y uno adicional para evitar la pérdida de datos durante las operaciones de estimación y borrado. Cada CS recibe las señales de control, el identificador de flujo y la señal que indica si el dato es válido. Las salidas del módulo incluyen la estimación de frecuencia usando 20 bits, la señal que propaga el *hash* correspondiente al flujo analizado de 32 bits y la señal que actúa como bandera de sincronización indicando que las señales de salida contienen datos válidos. La implementación de cada CS sigue la estructura de la Figura 3.2, como se describió en la Sección 4.5.1.

Las frecuencias estimadas de los diez CS activos en cada ventana se suman para obtener las estimaciones de la ventana, como se muestra en la Figura 5.5. Para ello, se implementó un árbol de reducción (o árbol de suma jerárquico) y un *pipeline* que sincroniza la señal que indica si una estimación es válida para el flujo actual y el *hash* con la suma final, para que los datos estén disponibles en el mismo ciclo de reloj para ser enviados al bloque siguiente, el `PQAs`. El árbol se estructuró en 4 etapas para sumar hasta once valores, considerando sólo los válidos. Se implementa un retardo de 3 ciclos sobre la señal de *hash* y un registro de desplazamiento controla la señal que indica si la salida de los CS es válida.

El bloque `Sketches HLL` estima la cardinalidad utilizando once *sketches* de forma cíclica, pero uno en cada ventana de tamaño W. Cada HLL recibe las señales de control, el identificador de flujo y la señal que indica si el dato es válido. Las salidas son el valor de la suma armónica de 32 bits y la señal que indica cuándo este valor es válido. La implementación de cada *sketch* sigue la estructura de la Figura 3.1 y es igual a la descrita en la Sección 4.5.1. Esta reutiliza un bloque de la función *MurmurHash3* del CS, junto con un bloque de memoria destinado al almacenamiento del *sketch*, un bloque que calcula la cantidad de ceros a la izquierda en los b bits del valor *hash* en tiempo logarítmico y un comparador. El acelerador utiliza una función *hash* de 32 bits con $p = 8$, $b = 24$ y cada *bucket* en tiene un ancho de 5 bits. La arquitectura diseñada permite el procesamiento de un nuevo identificador de flujo en cada ciclo de reloj.

El bloque `Contadores` incluye once contadores de 32 bits cada uno. Cada contador se inicializa en cero con un *reset* e incrementa únicamente cuando se recibe un paquete válido y la señal de control lo habilita. Esta lógica permite contabilizar el número de paquetes en cada ventana.

El bloque `PQAs` utiliza once PQA y cada una utiliza S bloques de memoria con R elementos almacenados, que se acceden en paralelo mediante un segundo bloque *MurmurHash3*, compartido también con el CS. Los $log_2 R$ bits menos significativos del *hash* se utilizan para indexar los S bloques de memoria. El resto de los bits del valor *hash* se almacenan en una etiqueta para identificar el flujo. El identificador se inserta con una frecuencia estimada. Un circuito combinacional compara la tupla etiqueta y estimación entrante con todas los elementos de la cola y actualiza su contenido según los resultados de esta comparación y conforme al procedimiento descrito en la Sección 3.4. Una máquina de estados maneja la transición entre el estado de recepción de elementos y el estado de lectura. Cada PQA se implementa de forma paralela y admite la inserción de un elemento por ciclo de reloj.

5.5.2. Módulo de estimación de entropía

El `módulo de estimación de entropía` calcula la entropía empírica de la ventana. Para ello, se leen todos los elementos del PQA correspondiente. El acelerador reduce el tiempo de ejecución leyendo simultáneamente los S bloques de salida del PQA, que entregan S tuplas que corresponden a los valores almacenados en una fila de la estructura. Cada uno de estos valores se inserta en uno de los S módulos de bloque `Ordenamiento Radix-Sort` que ordenan las columnas de cada PQA en paralelo. Cada módulo *Radix-Sort* entrega los elementos de la columna ordenados de mayor a menor, uno en cada ciclo de reloj. El funcionamiento e implementación del algoritmo *Radix-Sort* se describió en la Sección 4.5.2.

Posteriormente, el bloque `Merge del ordenamiento` lee los datos proporcionados por los módulos *Radix-Sort* y los guarda en S memorias independientes. Estas memorias son fusionadas o unidas de manera ordenada para generar una memoria que contenga los elementos de todas las columnas ordenados de mayor a menor. Para la unión se utiliza una estructura que almacena los punteros de lectura para cada una de las memorias de entrada, los cuales están inicializados en 0. Usando estos punteros, se lee el elemento correspondiente de cada memoria. Los elementos leídos se comparan usando un árbol, el cual determina cuál es el más grande y de qué memoria fue leído. El elemento más grande se escribe en la memoria de salida. Además, se incrementa el puntero de lectura de la memoria a la que corresponde el elemento seleccionado. En cada comparación se verifica que el puntero de lectura apunte a una posición válida de la memoria. El

proceso continúa hasta que se hayan escrito K elementos en la memoria; es decir, las frecuencias de los top-K flujos ordenados de forma descendente.

Una máquina de estados controla la escritura secuencial de datos en una memoria, una AXI-BRAM. En el estado denominado *Espera* el sistema espera la señal de habilitación, que indica la disponibilidad de datos ordenados; al activarse esta señal se transiciona al estado *Escritura*. Además, se inicializan en cero las señales asociadas al proceso de escritura como *addrb*: la dirección donde se escriben los dato para comenzar desde la primera posición de memoria; *wenb*: la señal de habilitación de escritura y *dinb*: el dato que se desea escribir en la memoria en la dirección especificada por *addrb*. En el estado *Escritura*, se activa *wenb* para permitir la escritura en la BRAM, se carga *dinb* con el valor de la frecuencia ordenado a almacenar y se incrementa addrb en cada ciclo de reloj para avanzar a la siguiente dirección. Una vez que *addrb* alcanza el límite superior, se retorna al estado *Espera*, reiniciando el proceso.

Las frecuencias de los top-K flujos son utilizadas para la estimación de los parámetros de la distribución de ley de potencias y la entropía empírica. Estos cálculos pueden ser implementadas en hardware, como se mostró en la Sección 4.5.2. Sin embargo, requieren operaciones de división y logaritmo que son complejas de implementar en hardware. Estos procesamientos no se hacen sobre datos en línea con una alta velocidad de llegada, sino sobre datos almacenados en memoria. Además, en cada desplazamiento, cada 6s se calcula solo una vez la entropía y la cardinalidad de la ventana, por lo que no es una tarea crítica en tiempo. Por lo tanto, estos cálculos se trasladaron al procesador AMD NEORV32 en esta implementación.

El bloque `NEORV32` se refiere al procesador IP NEORV32 [115] soft-core que puede implementarse en FPGAs y programarse en C. NEORV32 admite de forma nativa aritmética de enteros, operaciones lógicas, flujo de control y multiplicación y división de enteros. Las operaciones de punto flotante (IEEE 754 de precisión simple y doble) están disponibles mediante extensiones de hardware opcionales o emulación de software. Para la comunicación, hay *wrappers* disponibles para interfaces estándar como la Interfaz Avanzada Extensible (*Advanced eXtensible Interface*, AXI4) para comunicarse con periféricos, aceleradores y memoria externa cuando sea necesario. En la arquitectura diseñada se utilizó NOERV32 configurado con 256 KB de memoria de instrucciones y 256 KB de memoria de datos. Además, se comunicó con registros AXI, donde se almacenó la media armónica y el número de paquetes de cada ventana, provenientes del HLL y de los contadores, respectivamente. Las top-K frecuencias ordenadas se almacenaron en una BRAM con controlador AXI, que se puede acceder desde el procesador.

El acelerador usa tres dominios de reloj diferentes, por lo que se maneja el cruce de dominios

usando el módulo Cloking wizard. Este bloque permite generar una red de señales de reloj derivadas de una fuente de entrada, ajustadas en frecuencia, fase y ciclo de trabajo. Además, se utilizan 4 etapas de registros de sincronización para pasar de un dominio de reloj a otro, de la salida del bloque PQAs al de ordenamiento. En el caso, de los registros y memoria AXI se escriben con el reloj del merge del ordenamiento y se leen con el reloj del procesador. Este control de frecuencias es esencial para cumplir con los requisitos temporales de cada subsistema y maximizar el rendimiento general del diseño.

5.6. Resultados

En los experimentos realizados se usaron ocho trazas, solo del conjunto de datos público: MawiLab [33], porque su duración aproximada es de 900s mayor que las de CAIDA de aproximadamente 60s. El conjunto MawiLab contienen trazas recolectadas desde monitores de alta velocidad en enlaces troncales comerciales, desde 2006 hasta la actualidad. Las trazas de MawiLab presentan varias anomalías de red etiquetadas usando detectores de anomalías [33, 104, 105, 106, 107]. Para la estimación de entropía empírica las trazas se dividieron en ventanas de tiempo traslapadas de tamaño $W = 60$s y deslizamiento $s = \frac{W}{10} = 0{,}6$s. La estimación considera como identificadores del flujo las direcciones IP de origen y destino, puertos de origen y destino y el protocolo. Los paquetes que tienen uno o más atributos faltantes son tratados como redundantes y el algoritmo los ignora [108]. En la Tabla 5.1 se resumen las trazas; mostrando los valores medios del número total de paquetes $\bar{M}$, los valores medios de la cardinalidad de los flujos $\bar{N}$ y los valores medios de la entropía normalizada exacta $\bar{H}$. El número de ventanas W_k se calcula como $W_k = \frac{P-W}{s} + 1$, donde P es la duración exacta de la traza. En la Tabla 5.1 también se presentan las desviaciones estándar σ de $\bar{M}$, $\bar{N}$ y $\bar{H}$. Las trazas contienen en promedio, entre 1 y aproximadamente 3 millones de flujos con una σ_N entre 27 mil y 100 mil flujos; y entre 2 millones y 6 millones de paquetes con una σ_M entre 215 mil y 590 mil.

Para realizar la evaluación del método, el algoritmo y su acelerador hardware en cada ventana de tiempo deslizante, primero se selecciona el número de frecuencias estimadas empíricamente K, el parámetro p de los *sketches* HLL, las dimensiones de los *sketches* de cuentas, y las dimensiones de los arreglos de cola de prioridad utilizados. En la Sección 5.6.1, se utilizan cuatro trazas seleccionadas aleatoriamente del conjunto de datos para elegir parámetros que funcionen bien en escenarios reales. En las Secciones 5.6.2 y 5.6.3 se evaluan los resultados del método, el algoritmo y el acelerador propuesto usando las ocho trazas y el error relativo promedio de estimación de la entropía empírica.

Tabla 5.1: Trazas de MawiLab utilizadas en los experimentos, donde $\bar{M}$, $\bar{N}$ y $\bar{H}$ representan los valores promedio del número total de paquetes, la cardinalidad de los flujos y la entropía normalizada exacta, respectivamente; y σ_M, σ_N y σ_H las desviaciones estándar correspondientes. El número de ventanas de cada traza es representado por W_k.

Trazas	$\bar{M}$	σ_M	$\bar{N}$	σ_N	$\bar{H}$	σ_H	W_k
Mawi-20231300	3,309,764.74	215,032.49	1,018,537.21	52,586.32	0.7253	0.0268	140
Mawi-20231400	4,532,386.46	585,673.32	1,127,711.02	97,870.69	0.6455	0.0298	141
Mawi-20221400	2,318,614.98	155,372.71	1,016,830.56	59,111.93	0.8010	0.0192	141
Mawi-20211400	2,512,293.16	221,123.37	890,892.72	40,423.38	0.7665	0.0350	141
Mawi-20201400	4,650,534.74	244,306.32	2,767,766.38	39,309.25	0.8088	0.0193	35
Mawi-20191400	4,158,277.67	420,207.06	2,172,906.09	39,244.15	0.7807	0.0332	129
Mawi-20181400	5,104,770.39	367,529.41	1,848,272.64	39,679.04	0.6919	0.0207	140
Mawi-20161400	6,320,676.03	358,123.39	1,834,238.28	27,872.89	0.7047	0.0278	141

5.6.1. Parámetros del algoritmo

Número de elementos Top-K

En los experimentos se usa el error relativo promedio de estimación de la entropía para evaluar el impacto del valor de K en el método propuesto en la Sección 4.3. El error relativo promedio se define como:

$$\bar{E}_{r\_PL} = \frac{1}{W_k} \sum_{i=1}^{W_k} \frac{|H_{norm} - \hat{H}_{norm\_PL}|}{H_{norm}} \times 100, \tag{5.3}$$

donde H_{norm} es la entropía normalizada exacta en cada ventana según lo definido por la Ecuación (4.4). $\hat{H}_{norm\_PL}$ es la estimación del método como se define en la Ecuación (4.10), pero con los valores exactos de cardinalidad y frecuencias de los elementos top-K en cada ventana.

En la Figura 5.6 se muestra el error de estimación en las cuatro trazas seleccionadas, en función de K, expresado en potencias de dos entre 4,096 y 32,768. La estimación utiliza el método con los valores exactos de N y m_i en cada ventana. El último grupo de barras muestra el promedio del $\bar{E}_{r\_PL}$ de las cuatro trazas. El error disminuye para valores mayores de K, ya que en esos casos se pueden estimar mejor los parámetros de la distribución de ley de potencias que se usa para calcular $\hat{H}_2$. Para $K \geq 8,192$, el valor promedio de $\bar{E}_{r\_PL}$ es menor al 0.2 % y su valor máximo es inferior al 0.4 %, que permite discriminar entre la entropía de tráfico normal y distintos tipos de ataques DoS y DDoS [18]. Por lo tanto, se selecciona $K = 8,192$ como un

buen compromiso entre el error de estimación y uso de memoria.

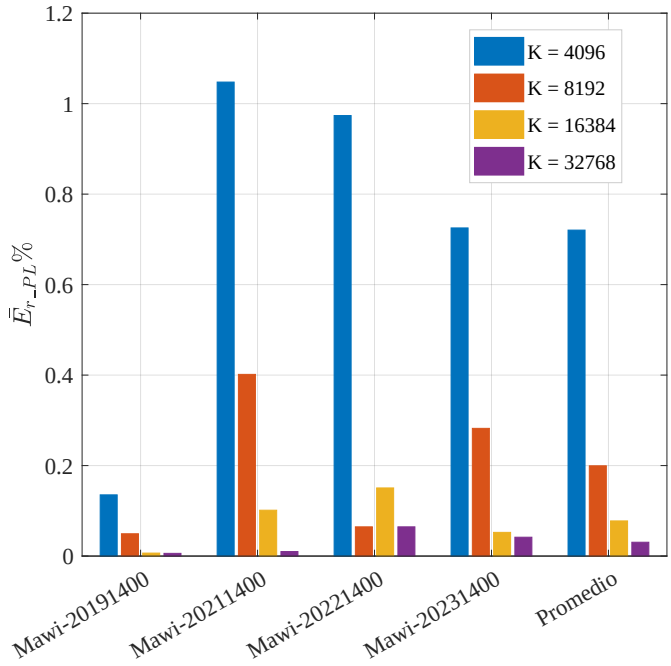


Figura 5.6: Errores promedio de estimación de la entropía empírica del método propuesto, relativo al valor exacto de la entropía calculada con la Ecuación (4.4), en función de K.

Parámetro de precisión de los *sketches* HLL

El segundo experimento evalúa el impacto del parámetro p del *sketch* HLL [29] en la estimación de entropía empírica, utilizando once *sketches* de igual dimensión como se describió en la Secciones 5.3 y 5.4. Para la evaluación se utiliza el error relativo promedio definido como:

$$\bar{E}_{r\_HLL} = \frac{1}{W_k} \sum_{i=1}^{W_k} \frac{|\hat{H}_{norm\_PL} - \hat{H}_{norm\_HLL}|}{\hat{H}_{norm\_PL}} \times 100, \tag{5.4}$$

donde $\hat{H}_{norm\_HLL}$ es la entropía normalizada calculada mediante la Ecuación (4.10) con $K = 8,192$, utilizando el valor de $\hat{N}$ estimado por el *sketch* HLL en cada ventana. La implementación de HLL utiliza la función *hash MurmurHash3* de 32 bits [93] y cada celda tiene un ancho de 5 bits. En la Figura 5.7 se muestra el error relativo promedio en la estimación de entropía del algoritmo en función del parámetro de precisión p, utilizando $p \in \{7, 8, 9, 10\}$. Para $p \geq 8$ se obtiene un $\bar{E}_{r\_HLL} < 0{,}34\,\%$ en las cuatro trazas, con un valor promedio inferior al $0{,}19\,\%$. Por lo tanto, se selecciona $p = 8$ para implementar el algoritmo.

Además, se compararon los algoritmos SHLL [75] y HLL uno en cada por ventana, utilizando

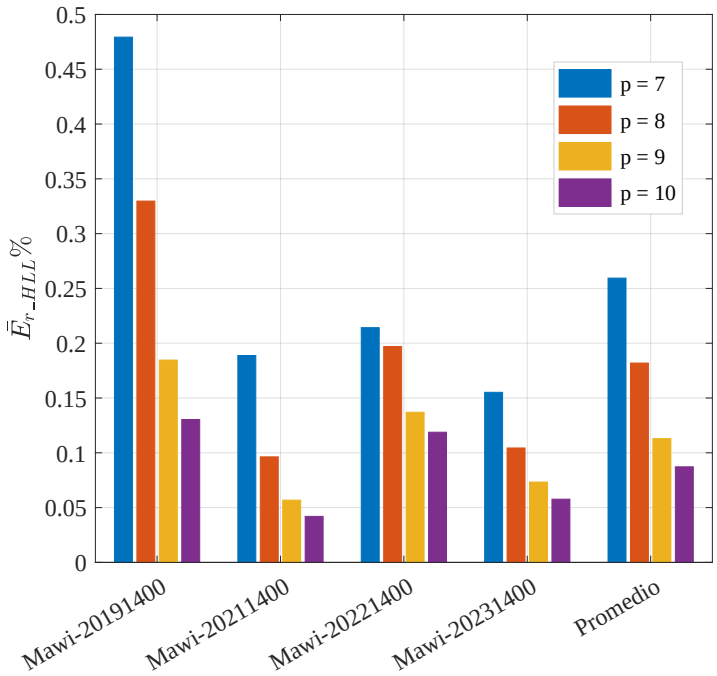


Figura 5.7: Errores promedio de estimación de entropía del algoritmo relativo a la entropía estimada mediante la Ecuación (4.10), en función del parámetro de precisión p de los *sketches* HLL.

el error relativo promedio en la estimación de cardinalidad, definido como:

$$\bar{E}_{r\_N} = \frac{1}{W_k} \sum_{i=1}^{W_k} \frac{|N - \hat{N}|}{N} \times 100, \tag{5.5}$$

donde $\hat{N}$ es la cardinalidad estimada y N es la cardinalidad real. En la Tabla 5.2 se muestran los resultados del $\bar{E}_{r\_N}$ para ambas estructuras con un uso de memoria similar, aproximadamente de 0.25 KiB. Para SHLL sus autores especifican un costo de memoria de $5mln(N/m)$ bytes y para once HLL $11mloglog(N/m)$ bits, donde $m = 2^p$ es el número de celdas en el *sketch*. Por lo tanto, considerando la cardinalidad exacta de las trazas, mostradas en la Tabla 5.1, se seleccionó $p = 2$ y $p = 8$ para SHLL y HLL, respectivamente. Los resultados muestran que el $\bar{E}_{r\_N}$ de SHLL es como mínimo seis veces mayor que con once HLL, debido a que el valor de p utilizado es menor. Su implementación en hardware es compleja, dado que requiere el almacenamiento y actualización de listas. La ventaja de SHLL es que permite estimaciones de cardinalidad de una ventana para cualquier deslizamiento y el uso de once HLL permite estimarla en ventanas de tamaño W y deslizamiento $s = \frac{W}{10}$. Sin embargo, este enfoque se ajusta al problema analizado con un buen compromiso entre precisión, uso de memoria y complejidad de implementación.

Tabla 5.2: Errores promedio en la estimación de cardinalidad de los algoritmo SHLL y HLL usando $p = 2$ y $p = 8$, respectivamente; relativo a la cardinalidad exacta N.

Trazas	$\bar{E}_{r\_N}(\%)$	
	SHLL	HLL
Mawi-20231400	62.4679	4.8299
Mawi-20221400	34.7216	5.1487
Mawi-20211400	87.6366	4.5696
Mawi-20191400	59.6196	5.6844
Promedio	**61.1115**	**5.0582**

Dimensiones de los *sketches* de cuentas

En el tercer experimento se evalúa el impacto de las dimensiones de los *sketches* de cuentas usados en cada intervalo s, para la estimación de entropía empírica. Se consideró el CS debido a su buen desempeño teórico y empírico en cuanto al error de estimación y a los resultados obtenidos en el capítulo anterior. El desempeño de los *sketches* se evaluó utilizando el error relativo promedio en la estimación de entropía respecto al método, definido como:

$$\bar{E}_{r\_s} = \frac{1}{W_k} \sum_{i=1}^{W_k} \frac{|\hat{H}_{norm\_PL} - \hat{H}_{norm\_s}|}{\hat{H}_{norm\_PL}} \times 100, \tag{5.6}$$

donde $\hat{H}_{norm\_s}$ es la entropía normalizada estimada mediante la Ecuación (4.10), utilizando las frecuencias estimadas por la unión de diez CS, la cardinalidad estimada $\hat{N}$ por un *sketch* HLL con $p = 8$ en cada ventana y $K = 8,192$.

En la Tabla 5.3 se muestran los resultados obtenidos. Cada fila de la tabla presenta el $\bar{E}_{r\_s}$ para cada uno de las cuatro trazas, considerando dimensiones del *sketch* como $w \in \{8192, 16384\}$ y $d \in \{7, 9\}$. La última fila de la tabla muestra el promedio de $\bar{E}_{r\_s}$ para las cuatro trazas del experimento. Los resultados evidencian que el error estimado producido por la unión de diez CS es más sensible al valor de w. Se resaltaron los resultados obtenidos para 16,384 $\times$ 9, que representan los menores errores de estimación y el promedio general del error es del 0.25 %.

En la Tabla 5.4 se muestra el error relativo promedio en la estimación de entropía usando saturación de los contadores, limitando los contadores del *sketch* al máximo valor que pueden representar según los bits especificados. El objetivo es reducir el uso de memoria del *sketch* manteniendo un error comparable a los mostrados en la Tabla 5.3 cuando $w = 16384$, $d = 9$ y se usan contadores de 32 bits. Los resultados muestran el error relativo promedio al saturar los

Tabla 5.3: Errores promedio en la estimación de la entropía, relativos a la entropía estimada mediante la Ecuación (4.10), usando las frecuencias estimadas por la unión de diez CS.

Trazas	w	$\bar{E}_{r\_s}(\%)$ Unión $CS \times s$	
		d =7	d = 9
Mawi-20231400	8K	2.4547	1.1403
	16K	0.1628	**0.0425**
Mawi-20221400	8K	2.0411	1.2038
	16K	0.1446	**0.1003**
Mawi-20211400	8K	2.1449	1.3831
	16K	0.6809	**0.4459**
Mawi-20191400	8K	2.3496	1.4301
	16K	0.6268	**0.4217**
Promedio	8K	2.2475	1.2893
	16K	0.4037	**0.2526**

contadores con 16, 18, 20, 22 y 24 bits. A partir de 20 bits el errror permanece constante e igual al obtenido con 32 bits. Por lo tanto, para la implementación del algoritmo se combinan 11 CS de $16,384 \times 9$ contadores de 20 bits cada uno.

Tabla 5.4: Errores promedio en la estimación de la entropía respecto a la entropía estimada mediante la Ecuación (4.10), usando las frecuencias estimadas por la unión de diez CS con $w = 16384$, $d = 9$ y saturación de contadores.

Trazas	$\bar{E}_{r\_s\_16K\times9}(\%)$					
	16 bits	18 bits	20 bits	22 bits	24 bits	32 bits
Mawi-20231400	0.8783	0.1217	0.0425	0.0425	0.0425	0.0425
Mawi-20221400	0.1003	0.1003	0.1003	0.1003	0.1003	0.1003
Mawi-20211400	1.8255	0.9159	0.4459	0.4459	0.4459	0.4459
Mawi-20191400	0.4934	0.4217	0.4217	0.4217	0.4217	0.4217
Promedio	**0.8244**	**0.3899**	**0.2529**	**0.2529**	**0.2529**	**0.2529**

En la Tabla 5.5 se muestran los errores relativos promedio en la estimación de la entropía cuando se utilizan diez CS, pero uno en cada ventana de tamaño W. En esta ventana es mayor el número de flujos distintos y sus frecuencias, por lo que se consideró $w \in \{16384, 32768\}$ y $d \in \{7, 9\}$. Los resultados muestran que para dimensiones de 16384×9 el promedio del $\bar{E}_{r\_s}$ es 2.4987 % comparable a 2.2475 % obtenido en la Tabla 5.3 cuando se combinan CS más pequeños

de $8,192 \times 7$ contadores. En la Tabla 5.5 cuando $w = 32,768$ y $d = 9$ se obtiene un promedio menor de 0.7202 %, para este caso los contadores también pueden saturarse en 20 bits. Para ello se analizó el máximo valor de las frecuencias para todas las ventanas y para cada traza.

Tabla 5.5: Errores promedio en la estimación de la entropía respecto a la entropía estimada mediante la Ecuación (4.10), usando las frecuencias estimadas por un CS en cada ventana W.

Trazas	w	$E_{r\_s}(\%)$	
		CS x W	
		d =7	$d = 9$
Mawi-20231400	16K	5.1945	**3.1036**
	32K	0.8858	0.2482
Mawi-20221400	16K	3.0822	**2.0508**
	32K	0.5693	0.1761
Mawi-20211400	16K	3.3522	**2.2837**
	32K	1.1896	0.8201
Mawi-20191400	16K	4.2447	**2.5566**
	32K	1.1196	0.6364
Promedio	16K	3.9684	**2.4987**
	32K	0.9411	0.7202

Además, se comparó el desempeño de *Unbiased Cleaning Sketch* (UC *Sketch*), analizado en la Sección 2.3.2 y la unión de diez CS. La comparación se realiza usando el error relativo promedio en la estimación de las frecuencias para todas las ventanas, definido como:

$$\bar{E}_{r\_m_i} = \frac{1}{KW_k} \sum_{i=1}^{W_k} \Big(\sum_{i=1}^{K} \frac{|m_i - \hat{m}_i|}{m_i} \Big) \times 100, \tag{5.7}$$

donde $\hat{m}_i$ es el conteo estimado de frecuencias y m_i es el conteo real para el flujo i-ésimo en cada ventana de tiempo deslizante. Dado que el método utiliza solo las frecuencias top-K para la estimación de parámetros de la ley de potencias, el error se calcula sólo para estos flujos.

En la Tabla 5.6 se presenta el error relativo promedio en la estimación de las frecuencias top-K $\bar{E}_{r\_m_i}$, utilizando los dos enfoques, con similar uso de memoria de aproximadamente 6 MiB. La segunda columna muestra los errores de UC *Sketch* usando la implementación en software proporcionada en [16]. Sus autores especificaron un uso de memoria igual a $4kb(d + 1) + 8kb$ bytes; donde k es el número de matrices, b es el número de estimadores o filas en una matriz y d es el número de columnas menos uno. Dada una ventana $W = 60$s y los parámetros sugeridos

por los autores se utilizó $k = 4$, $b = 32,768$ y $d = 9$. Sus resultados muestran para las cuatro trazas un promedio del $\bar{E}_{r\_m_i}$ aproximadamente del 66 % y un máximo aproximadamente del 97 %. En la tercera columna se considera la unión de diez CS, uno en cada intervalo s, todos con $16,384 \times 9$ contadores de 32 bits. Los resultados muestran una mejor estimación de frecuencias respecto a UC *Sketch* con un promedio del error relativo aproximadamente del 20 % y un valor máximo aproximadamente del 32 %. Aunque UC *Sketch* permite estimar en ventanas de diferentes dimensiones y deslizamientos a diferencia del enfoque de combinar *sketches*, sus errores para la memoria especificada y complejidad de implementación en FPGA es mayor, dado que requiere el trabajo con punteros y operaciones de escaneo.

Tabla 5.6: Errores promedio en la estimación de las frecuencias top-K utilizando UC *Sketch* y la unión de diez CS, relativos a los valores reales de frecuencias.

Trazas	$\bar{E}_{r\_m_i}$(%)	
	UC *Sketch*	Unión $CS \times s$
Mawi-20231400	97.18	26.35
Mawi-20221400	42.39	32.12
Mawi-20211400	74.77	10.24
Mawi-20191400	48.45	12.06
Promedio	**65.70**	**20.19**

Dimensiones del PQA

En el experimento anterior se consideró una cola de prioridad convencional perfecta en cada ventana, donde se almacenaban las frcuencias top-K estimadas por la unión de las estimaciones de frecuencias de diez CS. En este experimento esa cola de prioridad es reemplazada por un arreglo de R colas de prioridad de S elementos cada una, tal que $R \times S = K$ y $S \ll K$, como se analizó en la Sección 3.4. Se consideran dimensiones del PQA como 1024×8, 2048×4, 4096×2 y 8192×1. Se evaluaron estas alternativas utilizando el error relativo promedio en la estimación de entropía, según lo definido en la Ecuación (5.8):

$$\bar{E}_{r\_PQA} = \frac{1}{W_k} \sum_{i=1}^{W_k} \frac{|\hat{H}_{norm\_PL} - \hat{H}_{norm\_PQA}|}{\hat{H}_{norm\_PL}} \times 100 \tag{5.8}$$

donde $\hat{H}_{norm\_PQA}$ es la entropía normalizada estimada mediante la Ecuación (4.10), usando una PQA de 8,192 elementos, las frecuencias estimadas $\hat{m}_i$ por la unión de diez CS con $w = 16,384$ y $d = 9$ y la cardinalidad estimada $\hat{N}$ por un *sketch* HLL con $p = 8$ en cada ventana.

En la Tabla 5.7 se muestra el $\bar{E}_{r\_PQA}$ para las cuatro trazas y las cuatro configuraciones de dimensiones del PQA. El error de estimación es generalmente más bajo cuando S aumenta. Para $S = 4$, se obtiene el menor promedio, inferior al 0.20 % en relación con la estimación del método dada por la Ecuación (4.10) y un valor máximo de 0.39 %. Por lo tanto, en la implementación se utilizan once PQA de 2048 colas de prioridad de 4 elementos cada una.

Tabla 5.7: Errores promedio en la estimación de la entropía, respecto a la entropía estimada mediante la Ecuación (4.10), usando diferentes dimensiones del PQA.

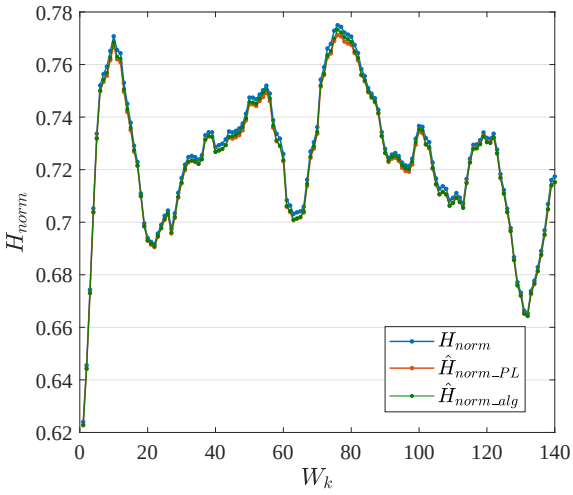
Trazas	$\bar{E}_{r\_PQA}$(%)			
	1024×8	2048×4	4096×2	8192×1
Mawi-20231400	0.0496	0.0639	0.1266	0.4728
Mawi-20221400	0.1096	0.1203	0.1492	0.3337
Mawi-20211400	0.3188	0.2229	0.1945	0.4348
Mawi-20191400	0.4039	0.3848	0.3299	0.1837
Promedio	**0.2205**	**0.1979**	**0.2001**	**0.3562**

5.6.2. Resultados del algoritmo

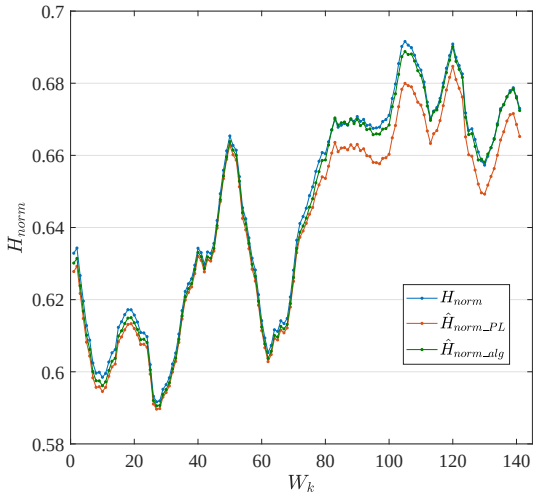
Error de estimación

En esta sección se examina la precisión de estimación del método y del algoritmo utilizando las 8 trazas de la Tabla 5.1. El algoritmo se implementó usando los parámetros seleccionados en la Sección 5.6.1: $K = 8,192$, once HLL con $p = 8$, once CS con $16,384 \times 9$ contadores de 20 bits y once PQA con $R = 2,048$ y $S = 4$.

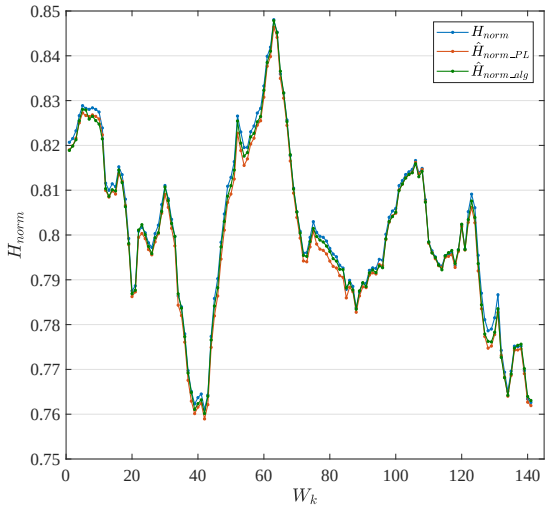
En la Tabla 5.8 se muestran los errores absoluto y relativo promedio del método propuesto ($\bar{E}_{abs}$ y $\bar{E}_r$) y del algoritmo ($\bar{E}_{abs\_alg}$ y $\bar{E}_{r\_alg}$), con respecto a la entropía empírica normalizada exacta. El valor promedio de $\bar{E}_{abs}$ y $\bar{E}_{abs\_alg}$ es aproxidamente 0.0022 tanto para el método como para el algoritmo, con valores máximos por debajo de 0.0054 y 0.0037, y desviaciones estándar de 0.0028 y 0.0012, respectivamente. El promedio del $\bar{E}_r$ es menor al 0.31 % y el promedio del $\bar{E}_{r\_alg}$ menor al 0.29 %, con valores máximos de 0.83 % y 0.46 %, y desviaciones estándar de 0.41 % y 0.15 %, respectivamente. El error promedio y máximo de estimación, tanto del método como del algoritmo, están muy por debajo del límite del 3 % que ha sido establecido como un umbral adecuado para discriminar entre tráfico de red normal y anómalo [35, 11]. El algoritmo produce resultados que se acercan a los del método propuesto, a pesar de las aproximaciones realizadas por las estructuras de datos utilizadas para reducir el uso de memoria.



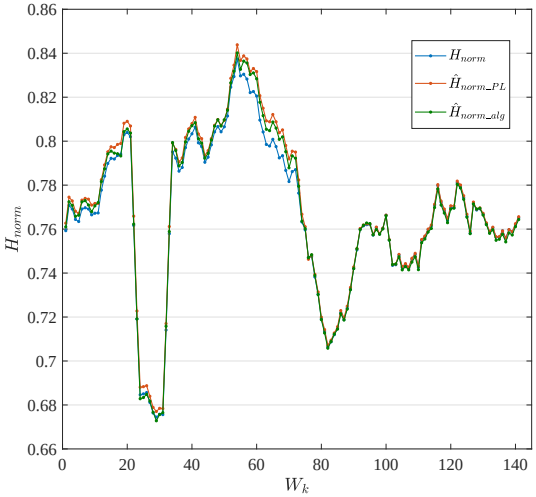
(a) Mawi-20231400



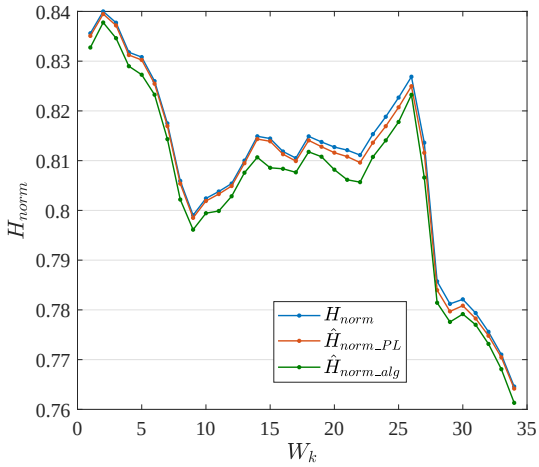
(b) Mawi-20231300



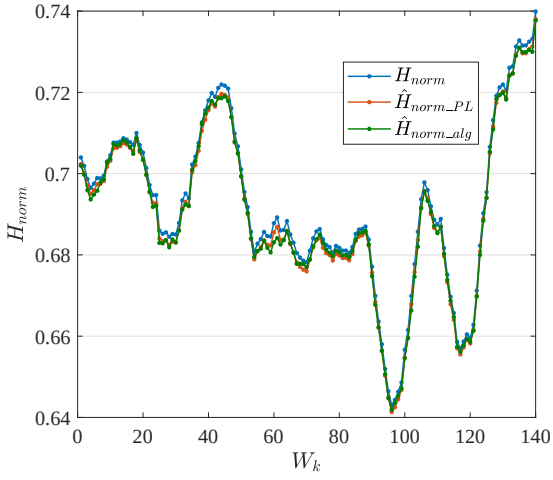
(c) Mawi-20221400



(d) Mawi-20211400



(e) Mawi-20201400



(f) Mawi-20181400

Figura 5.8: Entropía exacta calculada con la Ecuación (4.4), junto con las estimaciones del método y el algoritmo propuesto, para seis trazas de la Tabla 5.1.

Tabla 5.8: Errores promedios absolutos y relativos de estimación obtenidos mediante la Ecuación (4.10) y el Algoritmo 2 y 3, usando la entropía exacta como referencia.

Trazas	$\bar{E}_{abs}$	$\sigma_{E_{abs\_PL}}$	$\bar{E}_{abs\_alg}$	$\sigma_{E_{abs\_alg}}$	$\bar{E}_r$ (%)	$\sigma_{E_{r\_PL}}$	$\bar{E}_{r\_alg}$ (%)	$\sigma_{E_{r\_alg}}$
Mawi-20231300	0.0054	0.0028	0.0016	0.0007	0.8262	0.4049	0.2441	0.1224
Mawi-20231400	0.0021	0.0007	0.0017	0.0004	0.2823	0.0884	0.2273	0.0610
Mawi-20221400	0.0018	0.0010	0.0010	0.0007	0.2212	0.1279	0.1284	0.0913
Mawi-20211400	0.0031	0.0033	0.0021	0.0023	0.4016	0.4045	0.2662	0.2841
Mawi-20201400	0.0009	0.0005	0.0037	0.0012	0.1140	0.0659	0.4516	0.1414
Mawi-20191400	0.0004	0.0002	0.0030	0.0013	0.0493	0.0215	0.3828	0.1648
Mawi-20181400	0.0018	0.0003	0.0018	0.0008	0.2662	0.0442	0.2594	0.1207
Mawi-20161400	0.0020	0.0001	0.0023	0.0006	0.2815	0.0278	0.3321	0.0914
Promedio	**0.0022**	**0.0011**	**0.0022**	**0.0010**	**0.3053**	**0.1481**	**0.2865**	**0.1346**

En la Figura 5.8 se comparan los resultados de estimación de la entropía normalizada en cada ventana, para seis trazas aleatorias. H_{norm} es la entropía exacta de la traza. $\hat{H}_{norm\_PL}$ es la estimación de la entropía calculada mediante el método de la Ecuación (4.10) con $K = 8,192$. $\hat{H}_{norm\_alg}$ es la estimación de la entropía calculada mediante los Algoritmos 2 y 3, el cual incluye las aproximaciones de los *sketches* y los arreglos de cola, además de las aproximaciones de la integral y la derivada utilizadas en la Sección 5.4. En la figura se observa el seguimiento que permiten realizar el método y el algoritmo cada 6s de la entropía normalizada de los flujos observados en los últimos 60s.

5.6.3. Rendimiento del acelerador

El acelerador hardware se implementó en el plano de datos utilizando una tarjeta AMD Alveo U55C, que contiene un FPGA Virtex XCU55 UltraScale+. El acelerador se diseñó en SystemVerilog a nivel de transferencia de registros (*Register-Transfer Level*, RTL) y se implementó con la herramienta de síntesis AMD Vivado, versión 2024.2. Esta versión noo incluye automáticamente soporte para la tarjeta Alveo U280, utilizada en el Capítulo anterior.

Al utilizar todos los elementos del PQA de cada ventana para calcular los parámetros de la ley de potencias, no se realizaron optimizaciones que influyeran en el error de estimación y los cálculos se realizaron en el procesador NEORV32. Por lo tanto, los errores de estimación de entropía absolutos y relativos del algoritmo y del acelerador, usando la entropía exacta como referencia, son los mismos.

Tabla 5.9: Utilización de recursos del FPGA por núcleo (kernel).

Recursos	LUT	Flip-flop	BRAM	DSP
Bloque de control	885	86	0	0
11-HLL	6,416	9,470	0	374
11-CS	45,083	66,008	990	1,166
11-PQA	8,263	9,023	132	0
Radix-Sort	3,228	731	48	0
Merge Radix-Sort	503	671	22	0
NEORV32	2,553	1,869	33	6
Módulos AXI	2,812	3,646	2	0
Total	**69,743**	**91,504**	**1,227**	**1,540**
Utilización (%)	**5.42**	**3.59**	**56.3**	**16.31**

El acelerador trabaja con tres dominios de reloj. La arquitectura del módulo de `procesamiento de paquetes en ventanas deslizantes` se diseñó para maximizar la frecuencia de reloj, y así maximizar la velocidad de línea a la que puede operar el acelerador. En la implementación propuesta, este módulo funciona a 310 MHz, lo que permite alcanzar una velocidad de línea superior a 158 Gbps en el peor caso de paquetes de 64 bytes. La arquitectura de los módulos: `Ordenamiento Radix-Sort` y `Merge del ordenamiento` alcanzan una frecuencia de reloj máxima de 160 MHz. El procesador NEORV32 opera a una frecuencia de 300 MHz. El procesador calcula los parámetros de ley de potencia, la cardinalidad y la entropía de la ventana en aproximadamente 2,21s, tiempo menor al deslizamiento de la ventana de 6s. Esta latencia se debe principalmente a las operaciones logarítmicas y exponenciales de punto flotante. Esta sobrecarga se puede mitigar empleando un procesador *hard-core* con una unidad de punto flotante potente o utilizando tablas de interpolación y aritmética de punto fijo.

En la Tabla 5.9 se muestra la utilización de recursos del FPGA para los bloques principales del acelerador. Sólo se utiliza un 3.59 % de los flip-flops disponibles. La utilización de recursos lógicos (LUT) y aritméticos (DSP) es de aproximadamente 5.42 % y 16.31 %. El uso de memoria en el chip es más elevado: 56.3 % de la BRAM. Esta memoria se usa principalmente para implementar: los CSs, la estructura del PQA y los buffers usados por el *Radix-Sort*. El acelerador deja aproximadamente la mitad del hardware del FPGA libre, permitiendo así implementar otras tareas de medición y control de red en paralelo.

Respecto al acelerador propuesto en el capítulo anterior, esta implementación logra menor frecuencia de reloj. Esta frecuencia está determinada por el camino crítico o la ruta más larga

de propagación de señales lógicas entre dos registros secuenciales (flip-flops) dentro del circuito. El camino critico de la implementación está en el CS, en la escritura de los buckets con el valor nuevo, particularmente en la actualización del registro que tiene la dirección donde se va a escribir en cada fila. Dado que el diseño utiliza varias BRAM el tiempo de ruteo es alto. Eso se podría mejorar añadiendo más registros y aumentando el largo del *pipeline*.

Finalmente, el consumo de potencia del acelerador implementado en el FPGA Alveo U55C está compuesto por una potencia estática de 3.533 W y una potencia dinámica de 11.159W, lo que da como resultado un consumo total aproximado de 14.692 W. Dado que el TDP del Alveo U55C puede superar los 225 W, el acelerador está utilizando aproximadamente apenas un 6.5 % de la capacidad térmica total del dispositivo.

5.7. Discusión

En este capítulo se propuso un algoritmo y un acelerador hardware para la estimación de entropía empírica de Shannon en ventanas de tiempo deslizantes, diseñado para capturar con mayor fidelidad la dinámica del tráfico de red. La solución integra *sketches* de cardinalidad y frecuencia junto con arreglos de colas de prioridad, logrando un error relativo promedio inferior al 0.45 %, con desviaciones estándar menores al 0.15 %. Además, el acelerador alcanzó un rendimiento de línea de 158 Gbps considerando paquetes de 64 bytes, evidenciando que el modelo puede operar a velocidades de línea con tráfico masivo y manteniendo un uso de memoria hasta la mitad respecto a replicar la versión discreta en cada ventana. Los resultados obtenidos permiten generalizar que el uso combinado de *sketches* y colas de prioridad en un esquema de ventanas deslizantes constituye una estrategia eficaz para equilibrar precisión, eficiencia en memoria y velocidad de procesamiento.

Capítulo 6. Conclusiones y trabajo futuro

En la presente tesis se propusieron dos algoritmos para estimar la entropía empírica de Shannon de flujos de red, en intervalos de tiempo discreto y en ventanas de tiempo deslizantes. Con el fin de reducir el uso de memoria, el método estima empíricamente las frecuencias de los top-K flujos y asume que la distribución de probabilidades del resto sigue una ley de potencias. Calcula los parámetros de la distribución utilizando las frecuencias medidas de los top-K y estima las frecuencias del resto de los flujos para calcular una estimación de la entropía empírica. Los algoritmos utilizan *sketches* para estimar las frecuencias y la cardinalidad de los flujos, una estructura PQA que aproxima el comportamiento de una cola de prioridad para capturar las frecuencias de los top-K y aproximaciones que permiten estimar la entropía con buena eficiencia en espacio y tiempo. Ambos algoritmos en trazas de red reales muestran errores en las estimaciones que han demostrado ser suficiente para discriminar entre la entropía de tráfico normal y distintos tipos de ataques DoS y DDoS [18], lo que permite el uso de las mediciones obtenidas en sistemas de detección de anomalías.

En un conjunto de 16 trazas de los repositorios públicos de CAIDA, MAWILab e IoT-23 se demostró que el primer algoritmo, en intervalo de tiempo discreto, igual a la duración de las trazas, estima la entropía de los flujos con un error máximo del 2.42 % y un error medio del 0.69 %, en relación con la entropía exacta de las trazas. Los *sketches*, el PQA y otras aproximaciones introducen un error relativo medio de solo 0.06 % en comparación con la estimación del método. Implementado en C++, el algoritmo presenta un uso máximo de memoria de 3.15 MiB, independientemente del tamaño de la traza. Esto representa hasta más de tres órdenes de magnitud menos memoria que la requerida para calcular la entropía exacta de las trazas, y es comparable a otros métodos de estimación de entropía que presentan un error de estimación significativamente mayor. La mayor parte del error de estimación puede atribuirse al método, más que al algoritmo de estimación. Además, se evidenció que no todas las trazas siguen estrictamente una distribución de ley de potencias, pero combinar esta suposición con la estimación de las frecuencias de los elementos top-K, donde más se desvían los datos de esta distribución permitió obtener errores bajos de entropía. Para la aceleración hardware del algoritmo se diseñó una arquitectura en FPGA. El acelerador implementa el algoritmo completamente en el plano de datos y su arquitectura está diseñada para maximizar el *throughput*, reducir la latencia de estimación y aprovechar el hardware aritmético limitado disponible en los dispositivos FPGA. El acelerador en hardware utiliza aproximaciones aritméticas que aumentan el error medio de estimación en sólo 0.07 %. Implementado en una tarjeta Xilinx Alveo U280, puede procesar

tráfico de red a más de 204 Gbps y calcular la estimación de entropía con una latencia de 16 μs.

En un conjunto de ocho trazas del repositorio público MAWILab se demostró que el algoritmo en ventanas de 60s y deslizamiento 6s estima la entropía de los flujos con un error promedio máximo de 0.45 % con una desviación estándar de 0.14 %, en relación con la entropía exacta de las trazas. Implementado en C++ y Matlab, el algoritmo presenta un uso de memoria aproximado teórico de 4.3 MiB, independientemente del tamaño de la traza. La mayor parte del error de estimación puede atribuirse al método, más que al algoritmo de estimación. El acelerador implementa el algoritmo completamente en el plano de datos y su arquitectura está diseñada para maximizar el *throughput* en el FPGA. Implementado en una tarjeta AMD Alveo U55C, puede procesar tráfico de red a más de 158 Gbps; y presenta un uso de recursos del 56.3 % de los recursos disponibles en la tarjeta, lo que permitiría integrar hardware de procesamiento adicional en la misma plataforma. El uso del procesador soft-core NEORV32 entrega en menor tiempo de desarrollo estimaciones precisas de la entropía empírica con una latencia de 2.21s. Además, brinda reconfigurabilidad y flexibidad en modificaciones futuras que puedan realizarse en el cálculo de parámetros del método o de la distribución de probabilidades de los flujos.

Para el trabajo futuro se plantea:

- Implementar un arreglo de colas de prioridad deslizantes que permita almacenar las frecuencias de los flujos top-K en cada ventana de tiempo deslizante, para reducir el uso de memoria del método actual.
- Integrar los aceleradores hardware en la plataforma NetFPGA, la cual es compatible con la tarjeta Alveo U280, así como la adaptación de los algoritmos para implementarlo en *switches* programables utilizando P4.
- Evaluar distintos algoritmos de clasificación que puedan utilizar las mediciones de entropía empírica obtenidas para detectar anomalías en el tráfico de red.

Anexo A. Publicaciones

Durante el desarrollo de esta tesis se realizaron las siguientes investigaciones:

- Yaime Fernández, Javier E. Soto, Sofia Vera, Y. Prieto, Cecilia Hernández, and Miguel Figueroa. A streaming algorithm and hardware accelerator to estimate the empirical entropy of network flows. *Computer Networks*, 237, 110035, 2023.

  Publicada en la revista *Computer Networks* en 2023 presenta el diseño de un algoritmo de *streaming* basado en *sketches* y su acelerador hardware para la estimación de la entropía empírica de Shannon de flujos de red, en un intervalo de tiempo discreto igual a la duración de las trazas utilizadas. Este trabajo incluyó los resultados presentados en el Capítulo 4.

- Yaime Fernández, Javier E. Soto, Carolina Gallardo-Pavesi, Cecilia Hernández, and Miguel Figueroa. High-performance FPGA accelerator for entropy estimation of network flows in sliding time windows. *Computer Communications*, 2025 (Enviada).

  Enviada a la revista *Computer Communications* propone un algoritmo de *streaming* basado en *sketches* y su acelerador hardware para la estimación de la entropía empírica de Shannon de flujos de red, en ventanas de tiempo deslizantes. Este trabajo incluyó los resultados presentados en el Capítulo 5.

Además, he colaborado en las siguientes publicaciones, relacionadas con la investigación:

- 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*, 9:85823–85838, 2021.

- Javier E Soto, Sofía Vera, Yaime Fernández, Daniel Yunge, Cecilia Hernández, and Miguel Figueroa. A sketch-based algorithm for network-flow entropy estimation on programmable switches using p4. In 2023 *26th Euromicro Conference on Digital System Design (DSD)*, pages 79–86. IEEE, 2023.

- Carolina Gallardo-Pavesi, Yaime Fernández, Javier E Soto, Cecilia Hernández, Miguel Figueroa. A hardware accelerator for quantile estimation of network packet attributes. In 2024 *27th Euromicro Conference on Digital System Design (DSD)*, pp. 114-121. IEEE, 2024.

- Carolina Gallardo-Pavesi, Yaime Fernández, Javier E Soto, Cecilia Hernández, and Miguel Figueroa. A streaming algorithm and hardware accelerator for top-K flow detection in network traffic. In 2025 *28th Euromicro Conference on Digital System Design (DSD)*, (Aceptada).
- Carolina Gallardo-Pavesi, Yaime Fernández, Javier E Soto, Miguel Durán, Cecilia Hernández, and Miguel Figueroa. Sketch-based algorithm and hardware accelerator for quantile estimation of network flow. *Computer Networks*, 2025, (Enviada).

Referencias bibliográficas

[1] Claude E Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

[2] Gilberto Fernandes, Joel JPC Rodrigues, Luiz Fernando Carvalho, Jalal F Al-Muhtadi, and Mario Lemes Proença. A comprehensive survey on network anomaly detection. *Telecommunication Systems*, 70:447–489, 2019.

[3] Ki-Soon Yu, Sung-Hyun Kim, Dae-Woon Lim, and Young-Sik Kim. A multiple Rényi entropy based intrusion detection system for connected vehicles. *Entropy*, 22(2):186, 2020.

[4] Asghar Zarei and Babak Mohammadzadeh Asl. Automatic seizure detection using orthogonal matching pursuit, discrete wavelet transform, and entropy based features of eeg signals. *Computers in Biology and Medicine*, 131:104250, 2021.

[5] Petre Caraiani and Alexandru Vasile Lazarec. Using entropy to evaluate the impact of monetary policy shocks on financial networks. *Entropy*, 23(11):1465, 2021.

[6] Sunny Behal and Krishan Kumar. Detection of ddos attacks and flash events using novel information theory metrics. *Computer Networks*, 116:96–110, 2017.

[7] Darsh Patel, Kathiravan Srinivasan, Chuan-Yu Chang, Takshi Gupta, and Aman Kataria. Network anomaly detection inside consumer networks—a hybrid approach. *Electronics*, 9(6):923, 2020.

[8] Raja Majid Ali Ujjan, Zeeshan Pervez, Keshav Dahal, Wajahat Ali Khan, Asad Masood Khattak, and Bashir Hayat. Entropy based features distribution for anti-ddos model in sdn. *Sustainability*, 13(3):1522, 2021.

[9] Kun Zhou, Wenyong Wang, Chenhuang Wu, and Teng Hu. Practical evaluation of encrypted traffic classification based on a combined method of entropy estimation and neural networks. *Etri Journal*, 42(3):311–323, 2020.

[10] Ashraf Mohammed Saeed, Zaid Alyafeai, and Ashraf Mahmoud. Network traffic classifications using gated recurrent units with weighted cross-entropy. In *2022 14th International Conference on Computational Intelligence and Communication Networks (CICN)*, pages 218–223. IEEE, 2022.

[11] Ashwin Lall, Vyas Sekar, Mitsunori Ogihara, Jun Xu, and Hui Zhang. Data streaming algorithms for estimating entropy of network traffic. *ACM SIGMETRICS Performance Evaluation Review*, 34(1):145–156, 2006.

[12] Peter Clifford and Ioana Cosma. A simple sketching algorithm for entropy estimation over streaming data. In *Artificial Intelligence and Statistics*, pages 196–206. PMLR, 2013.

[13] Zijie Zeng, Lin Cui, Mimi Qian, Zhen Zhang, and Kaimin Wei. A survey on sliding window sketch for network measurement. *Computer Networks*, 226:109696, 2023.

[14] Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows. *SIAM journal on computing*, 31(6):1794–1813, 2002.

[15] Josep Sanjuas-Cuxart, Pere Barlet-Ros, and Josep Solé-Pareta. Counting flows over sliding windows in high speed networks. In *NETWORKING 2009: 8th International IFIP-TC 6 Networking Conference, Aachen, Germany, May 11-15, 2009. Proceedings 8*, pages 79–91. Springer, 2009.

[16] Yuhan Wu, Shiqi Jiang, Yifei Xu, Siyuan Dong, Kaicheng Yang, Peiqing Chen, and Tong Yang. Unbiased real-time traffic sketching. *IEEE Transactions on Network Science and Engineering*, 11(3):2371–2383, 2023.

[17] Somayeh Kianpisheh and Tarik Taleb. A survey on in-network computing: Programmable data plane and technology specific applications. *IEEE Communications Surveys & Tutorials*, 25(1):701–761, 2022.

[18] 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.

[19] 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.

[20] 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.

[21] He Huang, Yu-E Sun, Chaoyi Ma, Shigang Chen, Yang Du, Haibo Wang, and Qingjun Xiao. Spread estimation with non-duplicate sampling in high-speed networks. *IEEE/ACM Transactions on Networking*, 29(5):2073–2086, 2021.

[22] Biagio Peccerillo, Mirco Mannino, Andrea Mondelli, and Sandro Bartolini. A survey on hardware accelerators: Taxonomy, trends, challenges, and perspectives. *Journal of Systems Architecture*, 129:102561, 2022.

[23] Lu Tang, Qun Huang, and Patrick PC Lee. Mv-sketch: A fast and compact invertible sketch for heavy flow detection in network data streams. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 2026–2034. IEEE, 2019.

[24] Hui Han, Xuyang Jing, Zheng Yan, and Witold Pedrycz. Extendedsketch+: Super host identification and network host trust evaluation with memory efficiency and high accuracy. *Information Fusion*, 92:300–312, 2023.

[25] Hui Han, Zheng Yan, Xuyang Jing, and Witold Pedrycz. Applications of sketches in network traffic measurement: A survey. *Information Fusion*, 82:58–85, 2022.

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

[27] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. *Theoretical Computer Science*, 312(1):3–15, 2004.

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

[29] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. *Discrete mathematics & theoretical computer science*, (Proceedings), 2007.

[30] Xuyang Jing, Hui Han, Zheng Yan, and Witold Pedrycz. Supersketch: A multi-dimensional reversible data structure for super host identification. *IEEE Transactions on Dependable and Secure Computing*, 19(4):2741–2754, 2021.

[31] 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.

[32] CAIDA. The caida ucsd anonymized internet traces. data retrieved from CAIDA, `https://www.caida.org/data/passive/passive_dataset.xml`.

[33] Romain Fontugne, Pierre Borgnat, Patrice Abry, and Kensuke Fukuda. Mawilab: Combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking. In *Proceedings of the 6th International COnference*, pages 1–12, 2010.

[34] 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*, 9:85823–85838, 2021.

[35] Damu Ding, Marco Savi, and Domenico Siracusa. Tracking normalized network traffic entropy to detect ddos attacks in p4. *IEEE Transactions on Dependable and Secure Computing*, 19(6):4019–4031, 2021.

[36] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, et al. Hpcc: High precision congestion control. In *Proceedings of the ACM special interest group on data communication*, pages 44–58. 2019.

[37] Aviel Glam, Maayan Hacohen, and Barak Farbman. Improved load-balancing routing algorithms in manet using node cardinality metric. In *2021 IEEE International Conference on Microwaves, Antennas, Communications and Electronic Systems (COMCAS)*, pages 1–6. IEEE, 2021.

[38] David P Woodruff. New algorithms for heavy hitters in data streams. *arXiv preprint arXiv:1603.01733*, 2016.

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

[40] Zhen Mo, Yan Qiao, Shigang Chen, and Tao Li. Highly compact virtual maximum likelihood sketches for counting big network data. In *2014 52nd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 1188–1195. IEEE, 2014.

[41] Tao Qin, Zhaoli Liu, Pinghui Wang, Shancang Li, Xiaohong Guan, and Lixin Gao. Symmetry degree measurement and its applications to anomaly detection. *IEEE transactions on information forensics and security*, 15:1040–1055, 2019.

[42] Yang Zhou, Tong Yang, Jie Jiang, Bin Cui, Minlan Yu, Xiaoming Li, and Steve Uhlig. Cold filter: A meta-framework for faster and more accurate stream processing. In *Proceedings of the 2018 International Conference on Management of Data*, pages 741–756, 2018.

[43] Shigang Chen, Min Chen, and Qingjun Xiao. *Traffic measurement for big network data*. Springer, 2017.

[44] Pinghui Wang, Peng Jia, Jing Tao, and Xiaohong Guan. Mining long-term stealthy user behaviors on high speed links. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 2051–2059. IEEE, 2018.

[45] Jorge Crichigno, Elie Kfoury, Elias Bou-Harb, Nasir Ghani, Yasmany Prieto, Christian Vega, J Pezoa, C Huang, and David Torres. A flow-based entropy characterization of a nated network and its application on intrusion detection. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2019.

[46] Jan Havrda and František Charvát. Quantification method of classification processes. concept of structural a-entropy. *Kybernetika*, 3(1):30–35, 1967.

[47] Alfréd Rényi. On measures of entropy and information. In *Proceedings of the fourth Berkeley symposium on mathematical statistics and probability, volume 1: contributions to the theory of statistics*, volume 4, pages 547–562. University of California Press, 1961.

[48] Bernhard Tellenbach, Martin Burkhart, Dominik Schatzmann, David Gugelmann, and Didier Sornette. Accurate network anomaly classification with generalized entropy metrics. *Computer Networks*, 55(15):3485–3502, 2011.

[49] Christian Callegari, Stefano Giordano, and Michele Pagano. An information-theoretic method for the detection of anomalies in network traffic. *Computers & Security*, 70:351–365, 2017.

[50] Teodora Komazec and Slavko Gajin. Analysis of flow-based anomaly detection using shannon's entropy. In *2019 27th Telecommunications Forum (TELFOR)*, pages 1–4. IEEE, 2019.

[51] Qun Huang, Xin Jin, Patrick PC Lee, Runhui Li, Lu Tang, Yi-Chao Chen, and Gong Zhang. Sketchvisor: Robust network measurement for software packet processing. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 113–126, 2017.

[52] 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, 2019.

[53] 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.

[54] Arish Sateesan, Jo Vliegen, Simon Scherrer, Hsu-Chun Hsiao, Adrian Perrig, and Nele Mentens. Speed records in network flow measurement on fpga. In *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*, pages 219–224. IEEE, 2021.

[55] Christian Callegari, Stefano Giordano, and Michele Pagano. On the combined use of sketches and cusum for anomaly detection. In *2015 International Conference on Computing and Network Communications (CoCoNet)*, pages 157–162. IEEE, 2015.

[56] Yu-Kuen Lai, Cheng-Lin Tsai, Cheng-Han Chuang, Xiu-Wen Ku, and Jim Hao Chen. Tabular interpolation approach based on stable random projection for estimating empirical entropy of high-speed network traffic. *IEEE Access*, 10:104934–104953, 2022.

[57] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 20–29, 1996.

[58] Tong Yang, Junzhi Gong, Haowei Zhang, Lei Zou, Lei Shi, and Xiaoming Li. Heavyguardian: Separate and guard hot items in data streams. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2584–2593, 2018.

[59] Xingguang Chen and Sibo Wang. Efficient approximate algorithms for empirical entropy and mutual information. In *Proceedings of the 2021 International Conference on Management of Data*, pages 274–286, 2021.

[60] Piotr Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *Journal of the ACM (JACM)*, 53(3):307–323, 2006.

[61] Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. One sketch to rule them all: Rethinking network flow monitoring with univmon. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 101–114, 2016.

[62] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. Elastic sketch: Adaptive and fast network-wide measurements. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 561–575, 2018.

[63] Yikai Zhao, Kaicheng Yang, Zirui Liu, Tong Yang, Li Chen, Shiyi Liu, Naiqian Zheng, Ruixin Wang, Hanbo Wu, Yi Wang, et al. {LightGuardian}: A {full-visibility}, lightweight, in-band telemetry system using sketchlets. In *18th USENIX symposium on networked systems design and implementation (NSDI 21)*, pages 991–1010, 2021.

[64] Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[65] Jintao He, Jiaqi Zhu, and Qun Huang. Histsketch: A compact data structure for accurate per-key distribution monitoring. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*, pages 2008–2021. IEEE, 2023.

[66] Eran Assaf, Ran Ben Basat, Gil Einziger, and Roy Friedman. Pay for a sliding bloom filter and get counting, distinct elements, and entropy for free. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 2204–2212. IEEE, 2018.

[67] Gil Einziger and Roy Friedman. Counting with tinytable: Every bit counts! In *Proceedings of the 17th International Conference on Distributed Computing and Networking*, pages 1–10, 2016.

[68] Alessandro Cornacchia, Giuseppe Bianchi, Andrea Bianco, and Paolo Giaccone. Staggered hll: Near-continuous-time cardinality estimation with no overhead. *Computer Communications*, 193:168–175, 2022.

[69] Nikita Ivkin, Ran Ben Basat, Zaoxing Liu, Gil Einziger, Roy Friedman, and Vladimir Braverman. I know what you did last summer: Network monitoring using interval queries. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 3(3):1–28, 2019.

[70] Ran Ben Basat, Roy Friedman, and Rana Shahout. Heavy hitters over interval queries. *arXiv preprint arXiv:1804.10740*, 2018.

[71] Vladimir Braverman and Rafail Ostrovsky. Smooth histograms for sliding windows. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*, pages 283–293. IEEE, 2007.

[72] Odysseas Papapetrou, Minos Garofalakis, and Antonios Deligiannakis. Sketch-based querying of distributed sliding-window data streams. *arXiv preprint arXiv:1207.0139*, 2012.

[73] Xiangyang Gou, Long He, Yinda Zhang, Ke Wang, Xilai Liu, Tong Yang, Yi Wang, and Bin Cui. Sliding sketches: A framework using time zones for data stream processing in sliding windows. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1015–1025, 2020.

[74] Zijun Hang, Yongjie Wang, and Yuliang Lu. Hss: A memory-efficient, accurate, and fast network measurement framework in sliding windows. *IEEE Transactions on Network and Service Management*, 2024.

[75] Yousra Chabchoub and Georges Heébrail. Sliding hyperloglog: Estimating cardinality in a data stream over a sliding window. In *2010 IEEE International Conference on Data Mining Workshops*, pages 1297–1303. IEEE, 2010.

[76] Keke Zheng, Wenzhu Chen, Botao Feng, and Hanxin Zhang. Detecting top-k flows combining probabilistic sketch and sliding window. *IEEE Access*, 2024.

[77] Alex Rousskov and Duane Wessels. High-performance benchmarking with web polygraph. *Software: Practice and Experience*, 34(2):187–211, 2004.

[78] Dataset for imc 2010 data center measurement, 2010. Available:`https://pages.cs.wisc.edu/tbenson/~IMC10_Data.html`.

[79] Ângelo Cardoso Lapolli, Jonatas Adilson Marques, and Luciano Paschoal Gaspary. Offloading real-time ddos attack detection to programmable data planes. In *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 19–27. IEEE, 2019.

[80] Alexandre da Silveira Ilha, Ângelo Cardoso Lapolli, Jonatas Adilson Marques, and Luciano Paschoal Gaspary. Euclid: A fully in-network, p4-based approach for real-time ddos attack detection and mitigation. *IEEE Transactions on Network and Service Management*, 18(3):3121–3139, 2020.

[81] Libardo Andrey Quintero González, Lucas Castanheira, Jonatas Adilson Marques, Alberto Schaeffer-Filho, and Luciano Paschoal Gaspary. Bungee: An adaptive pushback mechanism for ddos detection and mitigation in p4 data planes. In *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 393–401. IEEE, 2021.

[82] Javier E Soto, Sofía Vera, Yaime Fernández, Daniel Yunge, Cecilia Hernández, and Miguel Figueroa. A sketch-based algorithm for network-flow entropy estimation on programmable switches using p4. In *2023 26th Euromicro Conference on Digital System Design (DSD)*, pages 79–86. IEEE, 2023.

[83] Stefan Heule, Marc Nunkesser, and Alexander Hall. Hyperloglog in practice: Algorithmic engineering of a state of the art cardinality estimation algorithm. In *Proceedings of the 16th International Conference on Extending Database Technology*, pages 683–692, 2013.

[84] Shurong Zhang, Tianyu Gao, and Junxing Zhang. Ddos detection based on sliding window entropy and decision tree with programmable switch. In *International Conference on Computational & Experimental Engineering and Sciences*, pages 1037–1049. Springer, 2023.

[85] Yu-Kuen Lai, Theophilus Wellem, and Hui-Ping You. Hardware-assisted estimation of entropy norm for high-speed network traffic. *Electronics Letters*, 50(24):1845–1847, 2014.

[86] Yu-Kuen Lai, Po-Yu Huang, Ho-Ping Lee, Cheng-Lin Tsai, Cheng-Sheng Chang, Manh Hung Nguyen, Yu-Jau Lin, Te-Lung Liu, and Jim Hao Chen. Real-time ddos attack detection using sketch-based entropy estimation on the netfpga sume platform. In *2020 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, pages 1566–1570. Ieee, 2020.

[87] 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.

[88] Boyu Zhang, He Huang, Yu-E Sun, Yang Du, and Dan Wang. Jigsaw-sketch: a fast and accurate algorithm for finding top-k elephant flows in high-speed networks. *Science China Information Sciences*, 67(4):142101, 2024.

[89] 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, 2003.

[90] Grigorios Chrysos, Odysseas Papapetrou, Dionisios Pnevmatikatos, Apostolos Dollas, and Minos Garofalakis. Data stream statistics over sliding windows: How to summarize 150 million updates per second on a single node. In *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*, pages 278–285. IEEE, 2019.

[91] Xiangyang Gou, Yinda Zhang, Zhoujing Hu, Long He, Ke Wang, Xilai Liu, Tong Yang, Yi Wang, and Bin Cui. A sketch framework for approximate data stream processing in sliding windows. *IEEE Transactions on Knowledge and Data Engineering*, 35(5):4411–4424, 2022.

[92] Lianhua Chi and Xingquan Zhu. Hashing techniques: A survey and taxonomy. *ACM Computing Surveys (Csur)*, 50(1):1–36, 2017.

[93] Austin Appleby. Smhasher & murmurhash, 2012. data retrieved from, `https://code.google.com/p/smhasher`.

[94] Yang Zhou, Peng Liu, Hao Jin, Tong Yang, Shoujiang Dang, and Xiaoming Li. One memory access sketch: a more accurate and faster sketch for per-flow measurement. In *GLOBECOM 2017-2017 IEEE Global Communications Conference*, pages 1–6. IEEE, 2017.

[95] 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, 2012.

[96] Sebastian Garcia, Agustin Parmisano, and Maria Jose Erquiaga. Iot-23: A labeled dataset with malicious and benign iot network traffic (version 1.0.0) [data set]. zenodo, 2020. data retrieved from `https://https://www.stratosphereips.org/datasets-iot23`.

[97] Albert-László Barabási. Network science. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 371(1987):20120375, 2013.

[98] Ruonan Zhu, Jiaqi Chen, and Bingyi Kang. Power law and dimension of the maximum value for belief distribution with the maximum deng entropy. *IEEE Access*, 8:47713–47719, 2020.

[99] HL Harper. The method of least squares and some alternatives. part i, ii, ii, iv, v, vi. *International Satistical Review*, 42:147–174, 1974.

[100] Michel L Goldstein, Steven A Morris, and Gary G Yen. Problems with fitting to the power-law distribution. *The European Physical Journal B-Condensed Matter and Complex Systems*, 41(2):255–258, 2004.

[101] Aaron Clauset, Cosma Rohilla Shalizi, and Mark EJ Newman. Power-law distributions in empirical data. *SIAM review*, 51(4):661–703, 2009.

[102] Daniel Bundala and Jakub Závodnỳ. Optimal sorting networks. In *International Conference on Language and Automata Theory and Applications*, pages 236–247. Springer, 2014.

[103] Zehra Yildiz, Musa Aydin, and Guray Yilmaz. Parallelization of bitonic sort and radix sort algorithms on many core gpus. In *2013 International conference on electronics, computer and computation (ICECCO)*, pages 326–329. IEEE, 2013.

[104] Guillaume Dewaele, Kensuke Fukuda, Pierre Borgnat, Patrice Abry, and Kenjiro Cho. Extracting hidden anomalies using sketch and non gaussian multiresolution statistical detection procedures. In *Proceedings of the 2007 workshop on Large scale attack defense*, pages 145–152, 2007.

[105] Romain Fontugne and Kensuke Fukuda. A hough-transform-based anomaly detector with an adaptive time interval. *ACM SIGAPP Applied Computing Review*, 11(3):41–51, 2011.

[106] Haakon Ringberg, Augustin Soule, Jennifer Rexford, and Christophe Diot. Sensitivity of pca for traffic anomaly detection. In *Proceedings of the 2007 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 109–120, 2007.

[107] Daniela Brauckhoff, Xenofontas Dimitropoulos, Arno Wagner, and Kavè Salamatian. Anomaly extraction in backbone networks using association rules. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement*, pages 28–34, 2009.

[108] Heather Lawrence, Uchenna Ezeobi, Orly Tauil, Jacob Nosal, Owen Redwood, Yanyan Zhuang, and Gedare Bloom. Cupid: A labeled dataset with pentesting for evaluation of network intrusion detection. *Journal of Systems Architecture*, 129:102621, 2022.

[109] 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.

[110] 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.

[111] Lu Jie, Chen Hongchang, Sun Penghao, Hu Tao, and Zhang Zhen. Ordersketch: An unbiased and fast sketch for frequency estimation of data streams. *Computer Networks*, 201:108563, 2021.

[112] Javier E Soto, Cecilia Hernández, and Miguel Figueroa. Jacc-fpga: A hardware accelerator for jaccard similarity estimation using fpgas in the cloud. *Future Generation Computer Systems*, 138:26–42, 2023.

[113] Antonio Saavedra, Hans Lehnert, Cecilia Hernández, Gonzalo Carvajal, and Miguel Figueroa. Mining discriminative k-mers in dna sequences using sketches and hardware acceleration. *IEEE Access*, 8:114715–114732, 2020.

[114] Roland N Mfondoum, Antoni Ivanov, Pavlina Koleva, Vladimir Poulkov, and Agata Manolova. Outlier detection in streaming data for telecommunications and industrial applications: A survey. *Electronics*, 13(16):3339, 2024.

[115] S Nolting et al. The neorv32 risc-v processor. *Fischer," The neorv32 risc-v processor*, 2022.