HiMon: Achieving Low-cost and High-accuracy Network Monitoring via Hierarchical Sketching

Yuchen Zhang, Zhenyu Wen, Shibo He, Xiang Chen, Chaojie Gu, Jiming Chen, *Fellow, IEEE*

***Abstract*—As data centers continue to expand in size and complexity, obtaining global traffic insights necessitates aggregating statistical data from numerous individual nodes, a process critical for effective network management. However, in data centers, existing approaches often rely on querying individual endpoint hosts to gather cluster-wide statistics, which introduces substantial latency and reduces efficiency, particularly in large-scale deployments.**

To address this issue, we propose HiMon, a cost-efficient and high-accurate distributed monitoring system for optimizing traffic aggregation. HiMon enables distributed nodes to perform real-time, flow-level statistical processing and report the data to a master node with minimal bandwidth consumption. The master node aggregates the collected data to construct a comprehensive global traffic view. To enable high-speed and high-precision per-packet processing on child nodes, we introduce MaxSketch. MaxSketch's data structure and update strategy allow it to accurately estimate child node traffic with minimal memory and computational overhead. For high-speed aggregation on the master node, we present PolySketch, which significantly boosts aggregation efficiency by delegating most computational tasks to the child nodes. Together, the hierarchical sketch structures of MaxSketch and PolySketch form the HiMon monitoring system.

Experimental evaluations demonstrate that HiMon surpasses baseline algorithms, achieving a 17-210× improvement in traffic processing efficiency, a 25-42× reduction in master node bandwidth consumption, and a 3.69-8.97× increase in accuracy.

***Index Terms*—Monitoring, Sketch, Data Center**

I. Introduction

Modern data centers house hundreds to thousands of servers, forming the backbone of cloud computing infrastructure. Tenants deploy their applications on these servers, generating massive traffic volumes, e.g., platforms like AWS manage over 100 million requests per second. To ensure consistent performance, cloud service providers actively monitor this traffic and manage cloud networks accordingly. By analyzing diverse network patterns, providers can identify heavy hitters [13], [26], [33], optimize resource scheduling [17], [23], and enhance cloud security [5], [7]. Thus, networking traffic monitoring is foundational for cloud network operation and management.

Existing data centers adopt a *count and aggregate* strategy for traffic monitoring. This involves two phases: (1) **counting**, where entry points (e.g., switches, servers) perform localized statistical analysis of traffic, and (2) **aggregating**, where results are fused to build a global network view. Monitoring frameworks differ in how they distribute these tasks across infrastructure, balancing performance (accuracy, latency) against resource costs (CPU, bandwidth).

Cloud services are deployed in data centers, but due to limitations in resources such as network bandwidth and storage capacity, few monitoring systems can comprehensively track all the traffic within a data center. Monitoring individual clusters or logical units offers a more efficient and feasible solution. However, even at the cluster level, challenges persist, as the system must handle hundreds or thousands of servers and traffic volumes on the order of terabytes.

Traditional traffic monitoring methods in data centers involve collecting and analyzing data at aggregation points, such as load balancers, to derive statistical metrics. This approach, however, places substantial strain on network components and lacks scalability, limiting its effectiveness in large-scale environments. Moreover, their visibility is inherently limited to traffic routed through them, creating blind spots in distributed architectures (e.g., microservices) where traffic bypasses the load balancer. In contrast, modern monitoring systems often collect data directly from end-hosts and aggregate it at master nodes for analysis [9], [16]. This approach significantly reduces the burden on individual components and improves overall system scalability. However, aggregating metrics across thousands of servers introduces real-time issues. The time of each round of traffic monitoring increases linearly with the network scale. For example, PathDump [28] queries each end-host every 200 ms and requires 2-3 seconds to gather information, which delays diagnostics and misses bursty traffic. Secondly, collecting traffic data on the end-hosts will put pressure on the CPU and bandwidth, as nodes must process and transmit data continuously. Traditional solutions like NetFlow [8] and sFlow [15] mitigate overhead via sampling, but this sacrifices accuracy for rare or short-lived traffic. Hardware and computational constraints further force trade-offs between sampling rates and measurement fidelity. Thus, achieving high-accuracy global traffic visibility in modern data centers without prohibitive cost or latency remains an unresolved challenge.

In this paper, we present HiMon (**Hi**erarchical **Mon**itoring), a lightweight distributed network traffic monitoring system that collects traffic information of each

This work was supported by the National Nature Science Foundation of China under Grant 62472387 and Zhejiang Provincial Natural Science Foundation of Major Program (Youth Original Project) under Grant LDQ24F020001 and 2025C02263. *(Corresponding authors: Zhenyu Wen)*

Yuchen Zhang, Shibo He, Chaojie Gu and Jiming Chen are with the College of Control Science and Engineering, Zhejiang University, Hangzhou, Zhejiang, 310027, China (e-mail: 12332065@zju.edu.cn; s18he@zju.edu.cn; gucj@zju.edu.cn; cjm@zju.edu.cn).

Zhenyu Wen is with the Institute of Cyberspace Security, Zhejiang University of Technology, Hangzhou, Zhejiang, 310023, China (e-mail: wenluke427@gmail.com).

Xiang Chen is with the College of Computer Science and Technology, Zhejiang University, Hangzhou, Zhejiang, 310027, China (e-mail: wasdnsxchen@gmail.com).

Jiming Chen is with Hangzhou Danzi University, Hangzhou, Zhejiang, 310018, China (e-mail: cjm@zju.edu.cn).

end-host in a distributed manner with high precision and achieves high-speed aggregation of collected data. HIMON exploits a lightweight data structure to enable full traffic processing and introduces a new counter design that improves measurement fidelity. Furthermore, it achieves high-speed aggregation at the master node, facilitating the real-time construction of a global traffic view. HIMON achieves this by addressing the following two challenges.

High-fidelity Large Flow Profiling without Additional Memory. Existing sketch approaches require allocating a certain size of memory for their counters. The size of the count impacts the accuracy of the sketch. Given that data center failures are often triggered by high-volume traffic, existing additional memory space (cost) trade for more accurate information about high-volume traffic. HIMON challenges this common practice by reallocating existing counter memory for large flows. We introduce MaxSketch, a sketch-based measurement algorithm designed for low-cost traffic analysis at each slave node. Given that data center failures are often triggered by high-volume traffic, MaxSketch allocates a dedicated area in the counter to precisely capture large flows information instead of allocating additional space and uses a more precise update strategy. To further optimize bandwidth utilization in distributed data center monitoring systems, the algorithm implements data compression by selectively discarding counters for certain small flows, significantly reducing the volume of reported data. This design achieves a balance between measurement accuracy and bandwidth efficiency. As a result, MaxSketch reduces statistical error by 72.9% to 88.9% and achieves 100% accuracy in heavy hitter detection under our experimental setting.

High-speed Counting Aggregation without Low Accuracy Loss. In data centers, reduced data volume enables faster transmission and processing speeds, thereby enhancing aggregation efficiency. However, this advantage comes at the cost of decreased overall statistical accuracy. HIMON addresses this trade-off by optimizing both data structures and aggregation procedures, maintaining high aggregation accuracy while achieving low memory consumption and high processing speed. Specifically, the system employs PolySketch for rapid aggregation of statistical reports from all node servers, utilizing a lightweight mapping mechanism to ensure real-time processing of large-scale data reports at the master node. Furthermore, to guarantee the accuracy of aggregated traffic statistics, PolySketch can be flexibly configured with varying sizes, maintaining a balance between memory usage and measurement precision. PolySketch maintains over 99.5% accuracy of heavy hitter detection after aggregation.

We implement HIMON and conduct experiments. The experimental results indicate that the configuration of HIMON significantly impacts statistical results. Increasing memory allocation at either the child or master nodes can substantially enhance measurement accuracy. However, this improvement comes at the cost of prolonged aggregation time and higher reporting bandwidth consumption. HIMON achieves low-cost, reducing the bandwidth usage of the master node by 25-42$\times$ and decreasing the traffic statistics processing time by 17-210$\times$ compared to baseline algorithms.

In summary, the main contributions of this study are as follows:

- We design HiMon, a cost-efficient and high-accurate distributed monitoring system that balances accuracy, bandwidth efficiency, and computational overhead.
- We propose MaxSketch, a novel sketch-based measurement algorithm that eliminates the need for additional memory allocation while accurately capturing large flow information.
- We introduce PolySketch, an efficient data aggregation mechanism that enables high-speed traffic monitoring while maintaining over 99.5% accuracy in heavy hitter detection.

This paper is arranged as follows. Section II summarizes the background and motivation. Section III provides the design overview of HIMON. Section IV and V detail the two core designs of HIMON. Section VI presents the implementation and evaluation. Finally, section VII summarizes this work.

II. Motivation and Related Work

A. Motivation

Efficient and accurate distributed network monitoring is essential for the reliable operation of modern data centers, where real-time performance is a critical requirement. These data centers often comprise thousands of servers per cluster, each cluster managed by a centralized controller. With traffic volumes reaching terabits per second (Tbps), monitoring systems must support high-throughput data processing and transmission efficiency. A main challenge is the timely aggregation of local traffic statistics at the master node to maintain consistent global visibility. This requires an effective and scalable data structure for rapid reporting. To address this, we adopt a lightweight sketch-based data structure. Sketches facilitate high-speed packet processing and enable significant data compression, thereby reducing the bandwidth overhead for transmitting statistics. This not only alleviates network congestion but also improves the responsiveness of the reporting mechanism, allowing the master node to quickly assimilate data from distributed sources. By integrating sketches, the monitoring system achieves improved scalability, efficiency, and real-time capability. In the following, we detail the sketch-based methods employed for distributed network monitoring and analyze their limitations.

In distributed monitoring, the master node collects statistical information from each slave node to generate a global view. This process typically involves two methods. The first method queries the statistical data from each slave node individually, followed by aggregation of the results. The second method first aggregates the statistical data and then performs the query.

Querying Multiple Statistical Results. This aggregation method involves collecting the statistical data from each node at the master node and querying them individually. However, as the number of monitoring nodes increases, both memory usage and query time grow linearly, making it difficult to meet real-time requirements. Using the CAIDA dataset, we generate numerous 1000KB CM-Sketches as the statistical data for each node. We then query these sketches on a single machine,

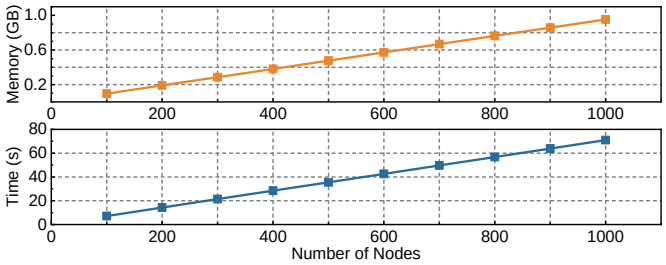


Fig. 1: The increase in memory and time overhead for storing or querying statistical information for each node.

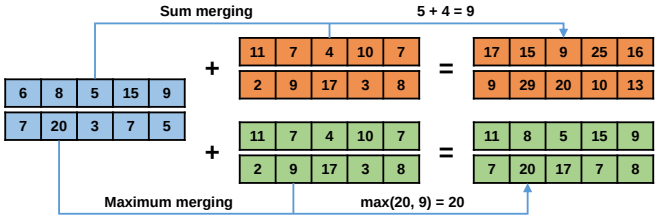
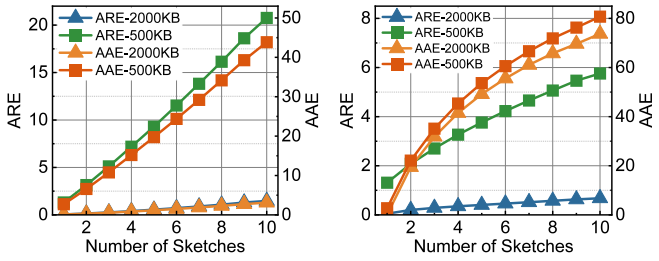


Fig. 2: Sum Merging of CM-Sketch



(a) Sum Merging (b) Maximum Merging

Fig. 3: The accuracy of CM-Sketch after sum or maximum merging.

recording memory usage and time consumption. As shown in Figure 1, both memory usage and query time increase with the number of sketches. When the number of nodes reaches 1,000, the query time exceeds one minute, and memory usage approaches 1 GB. Directly querying statistical data from each slave node, rather than collecting it at the master node, can substantially reduce memory usage for storing the statistics. However, this approach introduces additional aggregation time due to communication delays. Therefore, aggregating statistical data is essential. By merging redundant information and compressing low-density data, we can significantly reduce both memory usage and query time at the master node.

Merging Statistical Results. The most straightforward sketch merging method maps counters from the same position in each sketch to the corresponding location in a merged structure, either by summing or selecting the maximum value. Figure 2 illustrates both strategies. Using the CAIDA dataset [1], we generate 10 traffic segments of 3 million packets each. These are processed using 500 KB and 2000 KB CM-Sketches, and the resulting sketches are aggregated via sum or maximum merging. Evaluation based on Absolute Relative Error (ARE) and Average Absolute Error (AAE) shows that, under 500 KB memory, estimation error increases linearly with data volume, causing significant overestimation (Figure 3a). Allocating more memory (e.g., 2000 KB) improves accuracy but raises per-node resource demands, hindering scalability. The Elastic Sketch proposes a maximum merging variant under the assumption that flow keys are largely distinct across sketches. However, real-world traffic rarely meets this assumption, and changes in the selection of fields that make up the flow key can further invalidate it, reducing robustness. Figure 3b demonstrates that memory increase fails to mitigate underestimation for large flows when key overlap occurs. In summary, sum merging heightens hash collision risks, impairing estimation accuracy, while maximum merging loses information about large flows. Thus, neither method is suitable for large-scale distributed environments.

B. Related Work

Given these limitations in basic aggregation and merging techniques, numerous studies have sought to improve the efficiency and accuracy of distributed monitoring through alternative strategies. The following section reviews related efforts in distributed monitoring, sketch algorithms, and distributed sketch architectures.

Distributed monitoring. Currently, distributed monitoring systems leveraging endpoint hosts typically utilize their abundant resources for event monitoring [6], [21], [28], [29]. However, this often results in high CPU utilization on the end-hosts, while requesting statistical data from these hosts introduces additional bandwidth and time overhead. HIMON addresses these challenges by employing lightweight data structures for high-speed per-packet monitoring, thereby requiring minimal bandwidth to report the complete statistical data efficiently.

Sketch Algorithm. Current sketch algorithms are typically designed with specialized data structures for specific application scenarios. FCM-Sketch [27] features a tree-based data structure that improves memory utilization in CM-Sketch, resulting in a high-resolution universal data structure. Elastic Sketch [31] uses partial memory to create a key-value structure, enabling more accurate statistics for large flows. HistSketch [14] consists of a histogram structure that addresses key distribution issues through sketches. MC-Sketch [20] is designed to retain as much information as possible from high-priority flows, while low-priority flows are relegated to the lower layer of CM-Sketch. However, most of these algorithms lack built-in aggregation capabilities. As a result, when deployed in distributed settings, aggregation can only be performed using sum or maximum merging methods. These approaches often lead to significant accuracy degradation, as discussed in Section II-A.

Distributed Sketch. In order to solve the problem of a single point of failure, a lot of work has been proposed to deploy sketches at different locations on a traffic path. Existing distributed sketch methods primarily aim at efficiently allocating monitoring tasks across multiple switches, encompassing

both traffic partitioning and sketch partitioning strategies. Yu et al. [32] employ the parity of flow key hash values to partition traffic measurement tasks between ingress and egress switches. Gu et al. [12] introduce the concept of distributed sketch and divide the sketch into segments along the traffic forwarding path, reducing resource usage per switch. Li et al. [19] propose a lightweight framework that splits sketches into segments and allocates them across forwarding paths. However, [12], [19], [32] focus only on sketch task allocation along a single path. They distribute sketches across different switches and rely on inter-switch collaboration to reduce the resource overhead of traffic measurement. These approaches primarily capture path-local traffic statistics and thus have limited ability to obtain a global view of network traffic. In contrast, the distributed sketch method presented in this study does not concentrate on traffic measurement along a single path. Instead, it considers traffic at each end host and employs a hierarchical sketch structure to efficiently aggregate global traffic, thereby providing a system-level perspective for traffic analysis.

III. HiMon Overview

In order to meet the demand for traffic measurement in high-speed networks of data centers, we have the following three goals.

G1. High Throughput. The system must achieve exceptionally high throughput to handle the substantial traffic demands in data centers. This is essential to ensure that the HiMon can process and analyze the vast amounts of data generated without introducing bottlenecks, which could compromise the performance and reliability of the network.

G2. Bandwidth Efficiency. The system should minimize bandwidth consumption for data reporting to ensure seamless deployment in data centers, where resource optimization is vital for cost efficiency and maintaining the scalability of the infrastructure.

G3. High-speed Aggregation. The system requires a high aggregation rate to efficiently process large volumes of statistical data uploaded concurrently from slave nodes. This capability is necessary to prevent delays during the aggregation process, enabling timely and accurate traffic analysis, which is vital for maintaining the operational integrity of data center networks.

To achieve these goals, we design HiMon, as shown in Figure 4, which consists of two main components: *MaxSketch* on the slave node and *PolySketch* on the master node.

MaxSketch on Slave Node (end-host). MaxSketch is deployed on each slave node that requires traffic statistics. It monitors the traffic passing through the node and reports the data to the master node. To achieve **G1**, we minimize the packet processing time. Classical sketch algorithm employs multiple hash functions to mitigate statistical errors due to hash collisions, but the hash computation is the most time-consuming part of packet processing. We minimize the usage of hash computations while ensuring the accuracy of traffic statistics in MaxSketch. For **G2**, our goal is to reduce the data volume reported by each slave node, thus decreasing the bandwidth usage at the master node. First, we optimize the

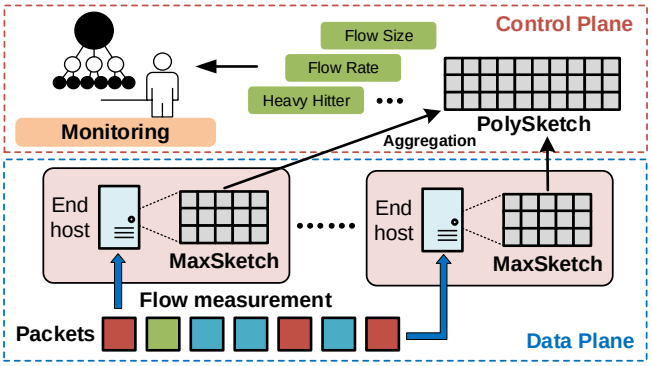


Fig. 4: HiMon design overview

MaxSketch data structure to enhance accuracy, enabling each slave node to use a smaller MaxSketch for traffic statistics. This optimization reduces the size of the reported sketch tables. Additionally, we apply a matrix compression algorithm to further reduce the size of the sketch tables, thereby minimizing the transmitted data volume and improving bandwidth efficiency. A detailed discussion of the MaxSketch design is presented in Section IV.

PolySketch on Master Node. PolySketch, deployed on the master node, aggregates global traffic statistics. Unlike MaxSketch, which independently monitors traffic, PolySketch aggregates MaxSketch data from all nodes. To achieve **G3**, we design a lightweight aggregation algorithm. We observe that decoding and merging traffic statistics from MaxSketch at the master node individually incur significant time overhead. To address this, we propose directly mapping the counters from MaxSketch to PolySketch using a hash-based method, eliminating the need for decoding. Furthermore, to reduce aggregation time, we offload the hash computations to MaxSketch, where they are performed before being reported to the central node for direct use. Details of the PolySketch design are in Section V.

IV. Distributed Measurement on End-Hosts

A. MaxSketch Design

Insight. In real-world network environments, traffic is predominantly occupied by a small number of large flows [2], [4], [10], [30]. Network operators often need to identify the top-k flows for network diagnostics, making the accurate estimation of large flows essential for network monitoring [3], [25]. To prevent small flows from interfering with the estimation of large flows, we allocate dedicated fields within each bucket to store statistics for large flows. Additionally, to minimize the impact of hash collisions, we store the flow key of large flows in the counters, enabling the filtering of conflicting flows. The flow key is hashed using an extra hash function into a two-byte space, which enhances memory efficiency. Furthermore, to ensure precise identification of large flows, we introduce an additional field to count packets involved in collisions.

Figure 5 shows the data structure of MaxSketch in one row, which consists of w buckets. Each bucket contains three types of fields: ID, PC, and NC, which occupy 2, 4, and 2 bytes,

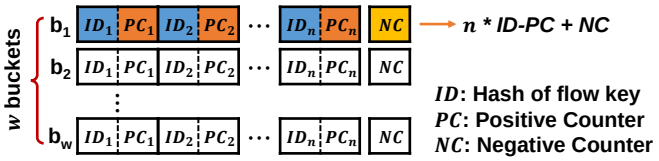


Fig. 5: Data structure of MaxSketch

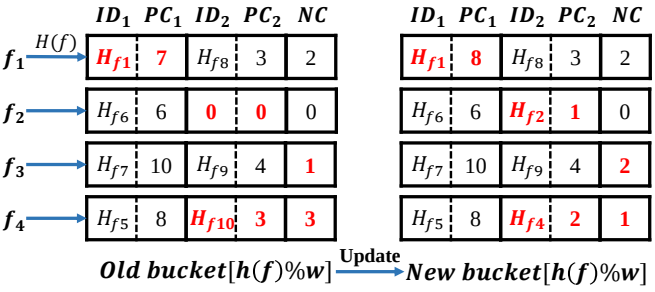


Fig. 6: Bucket update of MaxSketch

respectively. A complete MaxSketch can contain multiple rows. The ID field stores the flow key corresponding to the flow recorded in the PC field. This ID field is also reused in the aggregation algorithm, which we discuss in Section V. Since the length of the flow key is variable, an additional hash function is used to hash the flow key into a 2-byte space. The PC field records the frequency of the flow corresponding to the ID, representing the largest flow in the bucket. To prevent counter overflow, a larger space is allocated to the PC field. Additional ID-PC pairs (e.g., ID_1-PC_1 to ID_n-PC_n) can be pre-configured in each bucket to mitigate the effects of hash collisions. Lastly, the NC field tracks the number of packets from flows that hash to the bucket but conflict with the ID, representing small flows within the bucket.

As shown in algorithm 1, upon the arrival of a packet from flow f, the following operations are executed: first, the sketch column index $h(f)\%w$ is computed based on the flow key to locate the corresponding bucket. Next, $H(f)$ is computed to generate a two-byte flow ID, denoted as H_f. After identifying the bucket storing the packet's statistics, the counter update follows one of four cases:

1) If any ID matches H_f, the corresponding PC is incremented by 1. (Fig. 6-f_1)
2) If no ID matches H_f and an empty pair exists, insert $(H_f, 1)$, where $ID = H_f$ and $PC = 1$. (Fig. 6-f_2)
3) If no ID matches H_f and no empty pairs are found, the NC is incremented by 1. (Fig. 6-f_3)
4) If NC exceeds the PC of any pair, the alternative flow is upgraded to the preferred flow, with PC initialized to 2, NC initialized to 1, and the ID updated to H_f. (Fig. 6-f_4)

Before initiating the exchange between PC and NC, we ensure that $PC \geq 1$ and $NC \geq 2$. Consequently, initializing PC to 2 and NC to 1 during the exchange does not lead to overestimation. This update strategy nearly guarantees no overestimation of large flows. The only situation that could cause overestimation arises if two flows collide within the same bucket, and their IDs also collide. This situation occurs when both hash computations result in conflicts, a scenario with a very low probability.

Algorithm 1 MaxSketch Update Algorithm

Input: flow key k
1: **for** $i = 1 \rightarrow d$ **do**
2: $j \leftarrow h_i(k)$
3: $id \leftarrow H_i(k)$
4: **if** any (ID, PC) tuple empty **then**
5: $ID_{i,j} \leftarrow id$
6: $PC_{i,j} \leftarrow 1$
7: **else if** any ID == id **then**
8: $PC_{i,j} \leftarrow PC_{i,j} + 1$
9: **else**
10: $NC_{i,j} \leftarrow NC_{i,j} + 1$
11: **if** any m s.t. $NC_{i,j} > PC_{i,j,m}$ **then**
12: $ID_{i,j,m} \leftarrow id$
13: $NC_{i,j} \leftarrow 1$
14: $PC_{i,j,m} \leftarrow 2$
15: **end if**
16: **end if**
17: **end for**

MaxSketch also supports query operations, which is shown in Algorithm 2. The process begins by calculating the column index hash value and ID hash value based on the flow key (Line 2-3). Next, the bucket containing the flow's statistics is identified for each row, and the flow ID is compared (Line 4). If the IDs match, the count from PC is retrieved; otherwise, the default value of 1 is used (Line 5-7). Finally, the maximum value from all rows is output as the estimated value for the flow (Line 10). For instance, as illustrated in Figure 7, to determine the flow size of f_1, we first calculate $h_i(f_1)$ using the hash functions of each row to locate the buckets and then calculate H_{f1} as flow ID. The H_{f1} matches the IDs in the first and second rows. Consequently, the PC counter values from these rows are retrieved, which are 10 and 12. The maximum value among these counters is selected as the estimated flow size. Therefore, the estimation of flow f_1 is 12.

Algorithm 2 MaxSketch Query Algorithm

Input: flow key k
Output: estimate $\hat{f}(k)$ of flow k
1: **for** $i = 1 \rightarrow d$ **do**
2: $j \leftarrow h_i(k)$
3: $id \leftarrow H_i(k)$
4: **if** any $ID_{i,j} == id$ **then**
5: $\hat{f}_i(k) \leftarrow PC_{i,j}$
6: **else**
7: $\hat{f}_i(k) \leftarrow 1$
8: **end if**
9: **end for**
10: $\hat{f}(k) \leftarrow max_{1 \leq i \leq d} \hat{f}_i(k)$
11: **return** $\hat{f}(k)$;

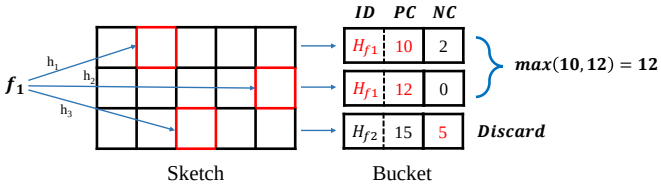


Fig. 7: Flow size query example of MaxSketch

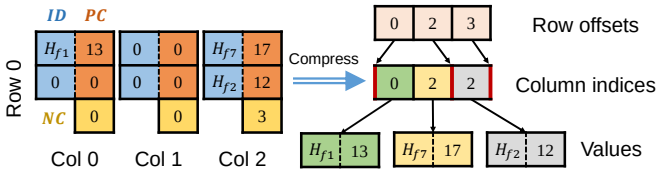


Fig. 8: Element mapping for MaxSKetch compression

B. MaxSketch Compression

Insight. Since the MaxSketch captures only a limited number of flows within each time window, a substantial portion of its buckets remain empty. At the master node, during the aggregation of sketches reported by slave nodes, every bucket must be traversed, and only nonzero data are processed. Consequently, empty buckets provide no valuable information during aggregation at the master node, making their upload redundant. To minimize the computational burden on the master node and reduce the data volume transmitted by slave nodes, a compression algorithm is applied to the MaxSketch. By employing the compression algorithm to offload the filtering of nonzero data to each slave node, the aggregation efficiency at the master node is significantly improved.

As shown in Figure 8, each sketch is compressed into three one-dimensional arrays, namely the row offset array, the column index array, and the value array. We consider multiple ID-PC pairs in a bucket as multiple rows. Therefore, the row offset array contains $d*p+1$ elements, where p is the number of ID-PC pairs in a bucket. In row offset array, each element r_i occupies 2 bytes, and represents the position of the first non-zero element in the $i-th$ row of the matrix in the column index array. Additionally, the last element stores the number N of non-zero elements in the entire matrix. The column index array contains a total of N elements. Each element c_i represents the column subscript of the value at the corresponding position in the value array. Combined with the row offset array, it can uniquely determine a position in the matrix. Finally, the value array also contains N elements and corresponds one by one to the column index array. Each element of the value array contains an ID-PC pair.

In data compression, the NC field is discarded. Because, during aggregation, each element is assigned to its corresponding block based on the ID field. Due to the absence of a corresponding ID record, the NC field has only a $1/n$ probability of being correctly assigned (see §V). Additionally, as the number of rows in the sketch is significantly smaller than the number of columns, the memory overhead of the row index array is negligible. Each ID-PC pair requires an additional 2-byte column index on top of its 6-byte data.

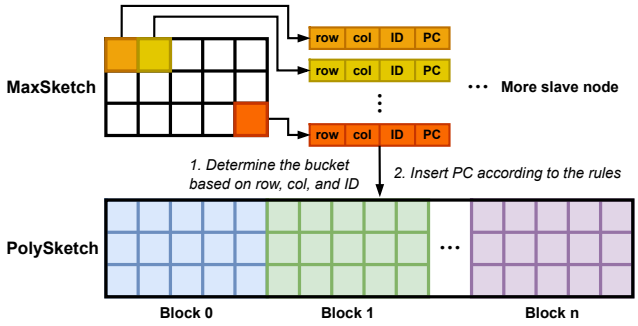


Fig. 9: Aggregation algorithm design

For example, in a MaxSketch containing two ID-PC pairs per bucket, the compressed memory usage becomes more efficient when the bucket occupancy rate falls below 87%. This condition is readily satisfied within a time window.

V. Aggregation Design

A. PolySketch on Master Node

Insight. Given the large number of servers in a data center, recording all the sketches from each server entirely would lead to significant memory and time overhead, along with poor scalability and reduced stability. For instance, in a medium-sized cluster with 1,000 servers, storing all sketches would require approximately 1 GB of memory, assuming each sketch consumes 1,000 KB. Furthermore, querying these sketches would require 1,000 times the time needed to extract statistical results from a single sketch. Therefore, we use a larger but fixed number of buckets to ensure the accuracy in aggregation and efficient memory usage. Considering that the aggregation efficiency is affected by the processing time of each uploaded sketch, and further affected by the processing situation of each element of the sketch, the insertion time of each element should be shortened as much as possible during the aggregation.

As shown in Figure 9, PolySketch, deployed on the master node, is structured as a sketch table with the same number of rows as MaxSketch but with n times the number of columns. Here, n is an adjustable integer that depends on the number of servers. During aggregation in each time window, n remains constant and does not vary with the number of servers, ensuring that aggregation can be performed with a fixed amount of memory. Each sketch table, which has the same size as MaxSketch, is referred to as a "block", meaning each PolySketch contains n blocks.

The core of the aggregation algorithm involves directly mapping the compressed data reported by each node to the corresponding counters in PolySketch, thereby eliminating the need for decoding and saving processing time. Additionally, the hash computations required for this mapping are offloaded to the ID field calculation in each MaxSketch. This approach not only distributes the computational load but also reduces the time required to process each entry. By using the ID value reported by the slave node, the block where the entry is stored

can be determined directly, enhancing the efficiency of the aggregation process.

Algorithm 3 Aggregation Algorithm

Input: row index i, column index j, value (id, pc)

```
b ← id % N                                  ▷ N is the number of blocks
if ID_{b,i,j} == id or ID_{b,i,j} == 0 then
    ID_{b,i,j} ← id
    PC_{b,i,j} ← PC_{b,i,j} + pc
else
    NC_{b,i,j} ← NC_{b,i,j} + pc
end if
if NC_{b,i,j} > PC_{b,i,j} then
    ID_{b,i,j} ← id
    PC_{b,i,j} ← pc
    NC_{b,i,j} ← 1
end if
```

The aggregation mainly includes the following steps. When the master node receives the sketch value sequence reported by the slave node, it traverses each value and insert them into PolySketch. Each reported value includes the row index, column index, ID hash value, and PC value. First, locate the block where the value is stored through the ID hash value, calculate the value of ID modulo n, and quickly obtain the block index. Secondly, locate the bucket used to summarize the statistical value in the block based on the row index and column index, and summarize the reported statistical value to this bucket. The update strategy of the PolySketch is similar to that of the MaxSketch (see Section IV-A). The key difference occurs in the fourth scenario ($NC > PC$). In this case, after updating the ID, the PC value in the corresponding bucket is set to the reported PC value, and the NC value is initialized to 1.

The PolySketch also supports query operations. First, the ID is computed based on the flow key. Similar to the update, the ID is then used to identify the block storing the flow. Subsequent query operations follow the same procedure as those of the MaxSketch, as described in Section IV-A.

VI. Evaluation

We evaluate the performance of HiMon using a tracking-driven approach. The experiment will be divided into the following parts.

- Evaluate the efficiency of traffic statistics and bandwidth usage to validate the advantages of HiMon over centralized monitoring (Experiments 1-1 and 1-2).
- Ablation experiments on the core configuration of HiMon and provide configuration recommendations (Experiments 2-1 to 2-3).
- Evaluate the aggregation rate of MaxSketch by PolySketch and analyze the accuracy of PolySketch in standard tasks (Experiments 3-1 to 3-6).
- Analyze the accuracy of MaxSketch in standard tasks and evaluate its throughput in packet processing (Experiments 4-1 to 4-4).

A. Setup

Testbed. We conduct tests on all algorithms using a virtual machine equipped with an AMD R7 2700X, which has 16 cores with 3.7 GHz clock speed, and 4GB of RAM running Ubuntu 22.04. We use two identical machines to deploy MaxSketch and PolySketch separately. Machine A processes the traffic and generates the sketch tables, which are then transmitted to machine B for aggregation, thereby constructing a distributed monitoring scenario. Unlike Internet devices, servers within data centers are centrally managed, allowing for reliable and efficient clock synchronization across all machines. Therefore, the evaluation assumes that clock synchronization of servers is already achieved and does not consider aggregation errors caused by time discrepancies. Since we do not consider the impact of data transmission channels in the experiment, MaxSketch can be deployed on a single machine. It processes data from different slave nodes in a time-division manner and then transmits the data to another machine for aggregation. In the following evaluation, we only use 1 core to process the traffic. Initially, we load the five-tuple information for all packets into memory to minimize file operation time.

Dataset. We use the CAIDA Internet Anonymized Traces dataset [1] collected on January 17, 2019. For small-scale experiments, we extract a 5-second segment containing approximately 3 million packets to evaluate single-node performance. For large-scale experiments, we employ the full one-hour dataset, dividing it into 1-second intervals to generate 3,500 traffic segments. This configuration simulates a system environment with up to 3,500 worker nodes. To address memory limitations, we compress the dataset by preserving only the five-tuple and timestamp information, which substantially reduces experimental memory overhead. Meanwhile, we use a key-value table to accurately count the actual number of packets for each flow, serving as the ground truth.

Evaluation Metrics. We consider the following metrics.

- Average Relative Error (ARE): $\frac{1}{n}\sum_{i=1}^{n}\frac{|f_i-\widehat{f}_i|}{f_i}$, where n is the total number of flows, f_i is the precise value of the flow, and $\widehat{f}_i$ is the estimated value. We use ARE to evaluate the accuracy of flow size estimation; however, due to the specificity of the algorithm in this paper, the estimated value is always less than the precise value, resulting in $ARE \leq 1$. Therefore, ARE cannot provide an accurate assessment when the error is large.
- Average Absolute Error (AAE): $\frac{1}{n}\sum_{i=1}^{n}\left|f_i-\widehat{f}_i\right|$. To address the limitations of ARE, we use AAE to assess the average packet deviation in flow size estimation.
- F1-score: $\frac{2\times PR\times RR}{PR+RR}$ where PR (Precision Rate) is the ratio of true instances reported and RR (Recall Rate) is the ratio of reported true instances. We use the F1-score to evaluate the accuracy of heavy hitter detection. We set 0.05% of the total traffic as the threshold, which means that flows with packet numbers exceeding this threshold will be marked as heavy hitters.
- Relative Error (RE):$\frac{|True-Estimated|}{True}$, which is used to evaluate the accuracy of entropy estimations.
- Throughput: packets per second (pps), flows per second (fps), and sketches per second (sps). We evaluate the

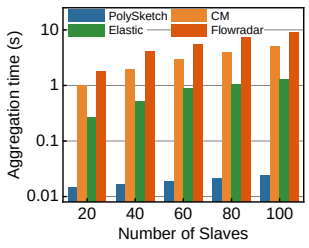


Fig. 10: Time cost on aggregating slave nodes.

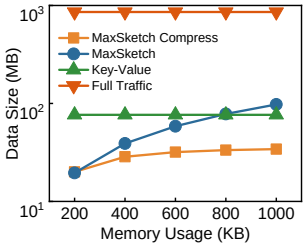


Fig. 11: Bandwidth cost of data reports on slave nodes.

insertion and query rates of the sketch using two different throughput metrics. We also use the third metric to evaluate the insertion rate of PolySketch.

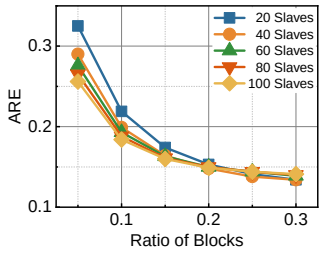
B. HiMon *Overall Evaluation*

In this subsection, we assess the overall performance of HiMon and compare it to centralized methods. The findings reveal that HiMon surpasses centralized approaches in both processing time and bandwidth resource usage.

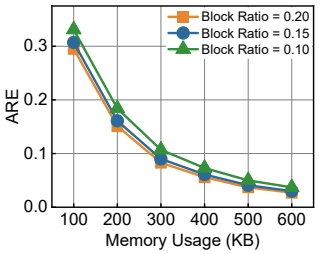
Experiment 1-1 (Traffic Statistics Efficiency). We compare the efficiency of centralized and distributed traffic statistics methods. The centralized algorithm processes all traffic on a single node, while the distributed algorithm distributes the traffic processing across multiple nodes. For centralized statistics, we utilize CM Sketch and Elastic Sketch. For HiMon, MaxSketch records the traffic, which is then aggregated into PolySketch. Multiple slave nodes, each handling approximately the same traffic volume, are configured to assess HiMon's performance under varying numbers of slave nodes. Finally, we measure the time taken by each method to complete the entire traffic processing, using this as the primary indicator of algorithm efficiency.

As shown in Figure 10, the statistical efficiency of HiMon significantly surpasses that of centralized methods. With 20 slave nodes, HiMon achieves 68 × and 18 × higher efficiency than Elastic and CM, respectively. With 100 slave nodes, the efficiency improvement increases to 210 × and 53 ×, respectively. As the number of slave nodes grows, the advantage of HiMon continues to expand.

Experiment 1-2 (Bandwidth Save). To assess the bandwidth savings at the master node with HiMon, we configure 100 slave nodes to perform traffic statistics and compression. We then calculate the total data volume reported to the master node. For comparison, we deploy non-compressed MaxSketch and key-value statistical methods on the slave nodes, using the total volume of raw traffic reported to the master node as the baseline. As illustrated in Figure 11, a 200 KB sketch size on the slave node renders the compression algorithm ineffective due to the high number of flows. As the MaxSketch size increased, the compression effect becomes more evident, and the reported data volume exhibits a sub-linear growth relative to memory usage per MaxSketch. With a 1000 KB MaxSketch, the compression rate achieved 35.0%, which is 65.0% less than the reported bandwidth before compression, and 55.6% less than the key-value data structure.



(a) Block Configuration



(b) Memory Configuration

Fig. 12: The impact of different configurations on the accuracy after aggregation. (a) Accuracy comparison under different block-to-node ratios. (b) Impact of MaxSketch and per-block memory configuration on aggregation accuracy.

C. HiMon *Configuration Evaluation*

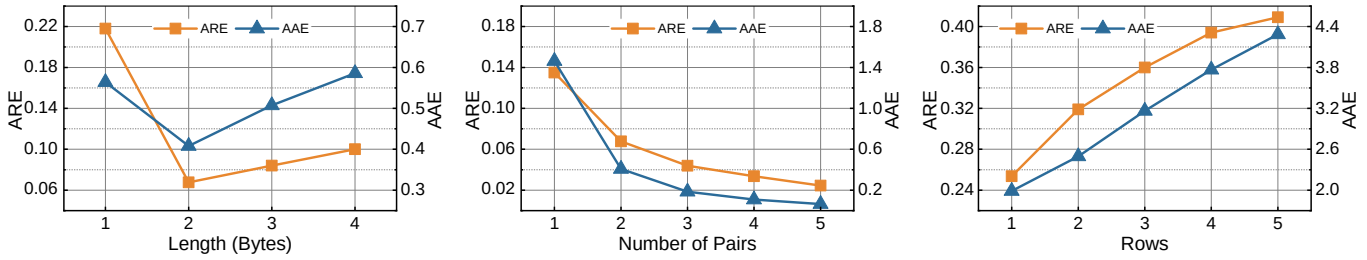
In this subsection, we assess the influence of various parameters on HiMon's performance through ablation experiments. The parameters considered include the block configuration of PolySketch, the bucket and row configurations of MaxSketch, and the time window configuration. Based on the results of these experiments, we offer several configuration recommendations.

Experiment 2-1 (Block Configuration). To determine the optimal number of blocks to configure in the PolySketch for varying slave node counts, we set the block count in the master node as a fixed proportion of the slave nodes. Each MaxSketch and block is configured to 200 KB. We compare accuracy variations across different slave node configurations with the same block-to-node ratio and analyze how accuracy changed as the number of slave nodes increased.

As shown in Figure 12a, we observe that increasing the proportion of blocks results in diminishing returns in accuracy improvement. When the block ratio increases from 0.05 to 0.1, accuracy improves by 29%. With a further increase to 0.15, accuracy improves by an additional 14%, but with a ratio of 0.2, the improvement was only 7%. Therefore, we recommend configuring blocks at 0.1 to 0.15 times the number of slave nodes, as this optimizes both memory usage and accuracy.

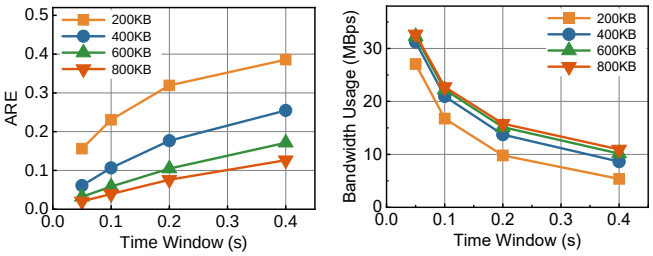
Next, we configure the block ratio to range from 0.1 to 0.2, fix the number of slave nodes at 100, and vary the MaxSketch size to assess the accuracy of the aggregated PolySketch. The results indicate a significant improvement in aggregated accuracy as the MaxSketch size increases. As illustrated in Figure 12b, for memory sizes below 400 KB, each 100 KB increment enhances accuracy by over 40%. At 500 KB memory size, the accuracy increases by 32.7%. Beyond 500 KB, accuracy improvement diminishes to less than 30%. Consequently, we recommend setting the MaxSketch size between 400 and 500 KB to achieve optimal performance under current traffic conditions.

Experiment 2-2 (MaxSketch Configuration). In MaxSketch, the ID field length is a critical configuration parameter that directly affects the ability of each bucket to handle hash collisions. To assess its impact, we evaluate flow size estimation accuracy using a 600 KB MaxSketch under different lengths. As shown in Figure 13a, a 2-byte ID field achieves the



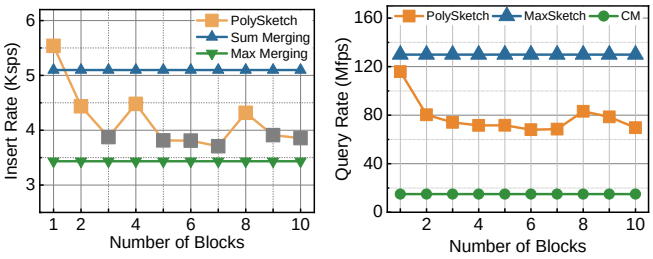
(a) ID Length Configuration (b) ID-PC Pairs Configuration (c) Row Configuration

Fig. 13: Accuracy comparison for different data structure configurations.



(a) ARE after Aggregation (b) Bandwidth Usage

Fig. 14: Accuracy comparison for different time window configurations on slave nodes.



(a) Aggregation (b) Query

Fig. 15: Throughput comparison of PolySketch.

highest estimation accuracy. Consequently, all experiments in this study adopt a 2-byte ID length as the default configuration.

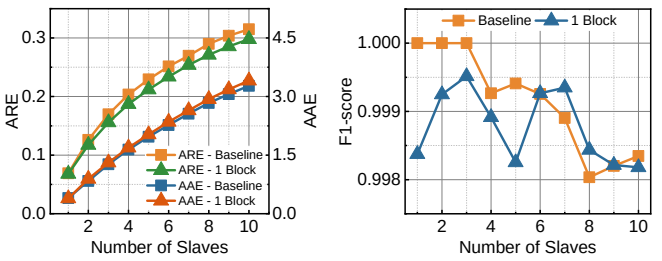
Furthermore, the number of ID-PC pairs in each bucket and the number of rows in MaxSketch will affect the accuracy. Figure 13b shows that increasing the number of ID-PC pairs in each bucket enhances accuracy while maintaining the same memory usage. The primary reason is that increasing the ID-PC pairs per bucket reduces the total number of buckets. Fewer buckets lead to fewer NC fields, as the NC fields have low information density, which ultimately enhances accuracy.

However, storing ID-PC pairs sequentially in each bucket increases access time as the number of pairs grows, which slows down insertion and query speeds. In an extreme scenario with only one bucket, accuracy reaches its maximum, but it incurs significantly high access costs for ID-PC pairs.

The figure illustrates that increasing the number of ID-PC pairs from 1 to 2 results in a 49.8% reduction in ARE and a 72.1% reduction in AAE, demonstrating a significant improvement in accuracy, while the insertion rate decreases by only 2.04%. When the number of ID-PC pairs increases from 2 to 3, the ARE decreases by 35.2% and the AAE decreases by 54.6%. Therefore, in practical applications, the HiMon data structure can be adjusted to meet specific requirements. If higher accuracy is needed, it is recommended to increase the number of ID-PC pairs, accepting a trade-off in throughput.

Figure 13c presents the ARE and AAE of HiMon under various configurations of rows. Increasing the number of rows in traditional sketches effectively mitigates hash collisions under fixed memory usage. However, adding rows decreases the number of columns, which can heighten the probability of hash collisions. Therefore, traditional sketches must balance the selection of rows and columns. For instance, CM-Sketch achieves optimal performance with three rows. In contrast, for HiMon, fewer rows result in higher accuracy because the data structure effectively addresses most hash collisions. Conversely, increasing the number of rows introduces additional redundant information, which decreases accuracy and increases hash computations, negatively impacting high-speed packet processing. In the heavy hitter detection task, increasing the number of rows does not substantially affect detection accuracy, as there are still sufficient buckets to accommodate candidate heavy hitters. However, increasing the number of rows raises the likelihood of hash collisions among stream IDs within the same bucket, resulting in some flows being overestimated and generating false positives.

Experiment 2-3 (Time Window). We evaluate the aggregation accuracy of PolySketch and the total reporting bandwidth of MaxSketch under various time window sizes, with a throughput of 6 Mpps. The experiment uses 10 slave nodes, with time window sizes ranging from 0.05 to 0.4 seconds. As shown in Figure 14, extending the time window increases the number of packets processed by each MaxSketch, leading to a higher rate of hash collisions. These collisions degrade the accuracy of MaxSketch's estimates, which in turn reduces the accuracy of the aggregated results in PolySketch. However, a larger time window also reduces the bandwidth required for reporting statistical data. This reduction occurs because more data is compressed within the same sketch, thus minimizing redundancy in the reported data. Therefore, it is essential to configure the time window size carefully, balancing accuracy, real-time requirements, and bandwidth limitations.



(a) Flow Size Estimation (b) Heavy Hitter Detection

Fig. 16: The error caused by the aggregation algorithms. Comparison of statistical results between aggregating multiple child nodes into a single block and the baseline (MaxSketch).

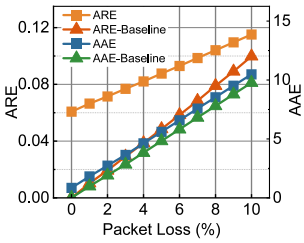


Fig. 17: Robustness after packet loss.

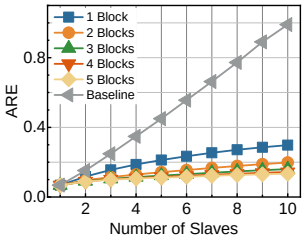


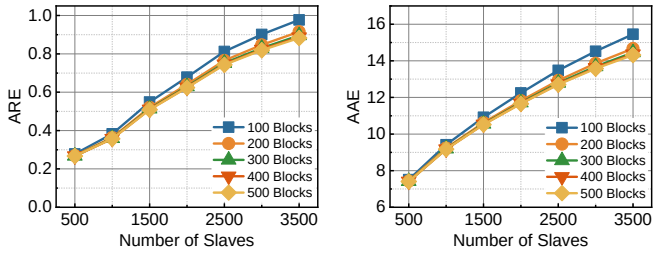
Fig. 18: Flow size estimation of aggregation.

D. PolySketch Performance Evaluation

In this subsection, we evaluate the performance of the aggregation algorithm in PolySketch. First, we assess the efficiency of PolySketch in aggregating MaxSketch data and examine its query rate. Next, we analyze the additional error introduced by the aggregation algorithm and its robustness in the presence of network failures. We also verify the impact of scaling the number of slave nodes on the accuracy of aggregation at the master node. Finally, we evaluate the performance of PolySketch on two classic tasks.

Experiment 3-1 (Throughput). As shown in Figure 15, we evaluate the aggregation and query rates of the PolySketch, comparing them with those of CM-Sketch using sum and maximum merging methods. In the figure, CM-Sketch and MaxSketch are represented by horizontal lines, as they do not have a block configuration, serving as baselines for comparison. We configure the MaxSketch to 200KB and use the number of sketches aggregated per second to evaluate the PolySketch's aggregation rate.

Figure 15a shows three distinct aggregation rate ranges: the highest at a block count of 1, followed by counts of 2, 4, and 8, with the lowest rates for other configurations. The PolySketch replaces hash calculations with a modulo operation on the ID by the block count during aggregation, significantly affecting the aggregation rate. A modulo operation with 1 always yields 0, resulting in maximum computational efficiency. A modulo operation with 2^n requires only a bitwise AND with $2^n - 1$, ensuring high computational efficiency. Conversely, modulo operations with other numbers are more computationally complex, reducing insertion rates. Thus, setting the block count to 2^n is recommended for optimal performance.



(a) ARE Comparison (b) AAE Comparison

Fig. 19: Accuracy of flow size estimation after slave node scale expansion.

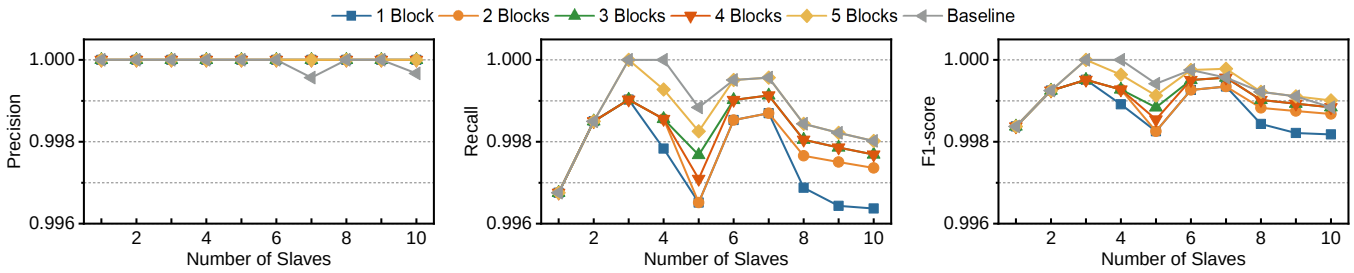
The aggregation rate of HIMON is approximately 20% lower than that of CM-Sketch using sum merging but nearly twice as high as that of maximum merging. The primary reason for this difference is that sum merging requires only one additional addition per counter traversal, which is close to the upper limit of the aggregation rate. In contrast, maximum merging requires more comparison operations, resulting in a slower rate.

The query rates for PolySketch and MaxSketch are nearly identical, as each flow involves two hash calculations. With a block count of 1, efficient modulo computation increases the query rate by about 3% compared to other configurations. Overall, the query rate of the PolySketch is 4.4% lower than that of the MaxSketch due to the additional modulus operation, yet still significantly higher than that of CM-Sketch.

Experiment 3-2 (Extra Error by Aggregation). We use 1 block of PolySketch to aggregate multiple MaxSketch, while the same size of memory is applied to directly process traffic. The errors from both methods are compared to assess the accuracy of the aggregation algorithm. Since every slave node utilizes memory equivalent to 1 block for statistics, comparing the aggregation result with more blocks to direct processing using the same memory is not meaningful in this experiment. As shown in Figure 16, the introduction of additional large-flow collisions leads to an average 6.8% increase in the aggregated AAE compared to the baseline. However, for the ARE metric, the error after aggregation decreases by an average of 3.0% compared to the baseline, indicating that the aggregation algorithm does not affect the statistical accuracy of small flows.

Experiment 3-3 (Robustness of Aggregation). In distributed measurement, each slave node reports its statistics to the master node. Due to the dynamic nature of data center networks, however, congestion may prevent some sketches from being reported promptly. Sketch packets not received within the current time window, whether due to congestion, clock skew, or other factors, are uniformly treated as packet loss events. In this part of the experiments, we evaluate their impact on PolySketch aggregation accuracy.

We use 100 slave nodes and randomly discard the data from 1 to 10 nodes. The error after aggregation is then calculated. Additionally, we compute the error between the accurate statistics of the remaining flows and the true values, which serves as the baseline.



(a) Precision (b) Recall (c) F1-score

Fig. 20: Heavy hitter detection accuracy comparison between different numbers of blocks.

As shown in Figure 17, the error increases with the packet loss rate. However, as the packet loss rate rises, the ARE gradually approaches the baseline, indicating that the master sketch's aggregation does not produce divergent errors due to packet loss. The AAE can be decomposed into two components: the error from the statistics and aggregation algorithms, and the error from packet loss. These two errors do not interfere with each other, so the AAE consistently remains at the same distance from the baseline, reflecting the algorithmic error.

Therefore, HIMON exhibits strong stability under packet loss, introducing minimal additional errors beyond those caused by the loss of statistical data.

Experiment 3-4 (Robustness of slave nodes expansion). The number of blocks, denoted by n, determines the maximum number of slave nodes that the master node can aggregate while maintaining acceptable accuracy. Generally, a single cluster comprises several thousand servers [22], [24]. Therefore, to assess how PolySketch performs when aggregating a large number of slave nodes, we extend it to 3,500 and vary n from 100 to 500. Experimental results show that, in the heavy hitter detection task, the F1-score consistently remains at 100%, demonstrating that the system exhibits strong scalability and tolerance to node expansion under a fixed block configuration. However, as shown in Figure 19, when the number of nodes continues to grow, conflicts within each block in PolySketch inevitably increase, leading to an approximately linear rise in ARE and AAE, and consequently, a gradual decline in estimation accuracy.

In practical deployments, the acceptable accuracy range can guide the determination of the maximum number of slave nodes that can be aggregated for a given block configuration. Since node scaling in data centers is typically predictable, the master node only needs to maintain sufficient robustness to accommodate short-term fluctuations in node count. Thus, the block number n can be reconfigured over relatively long intervals, ensuring stable memory usage at the master node while maintaining aggregation accuracy.

Experiment 3-5 (Flow Size Estimation of PolySketch). Figure 18 shows that aggregation algorithms significantly reduce the ARE, with accuracy improving as the number of blocks at the PolySketch increases. Using 1 block reduces the ARE by up to 69.9% and an average of 48.2% compared to the baseline. Using 5 blocks reduces the ARE by up to 86.5% and an average of 66.3% compared to the baseline. The high baseline ARE primarily arises from the potential overestimation of non-existent flows in each sketch, leading to a significant overestimation of many small flows. During aggregation, identical flows from different sketches are combined into a single counter, greatly reducing the likelihood of overestimating small flows.

Besides comparing with the baseline, we can also evaluate the sum merging method. The merged sketch is created by summing counters at corresponding positions of the original sketches. Notably, the merging of counters in HIMON must adhere to its specific update rules. The aggregated sketch generated by this method matches the PolySketch when the block is set to 1.

The curve trend indicates that as the number of sketches increases, the increase in ARE with the aggregation algorithm gradually slows, demonstrating its high robustness. Even with limited memory, aggregating a large number of sketches can still achieve high accuracy.

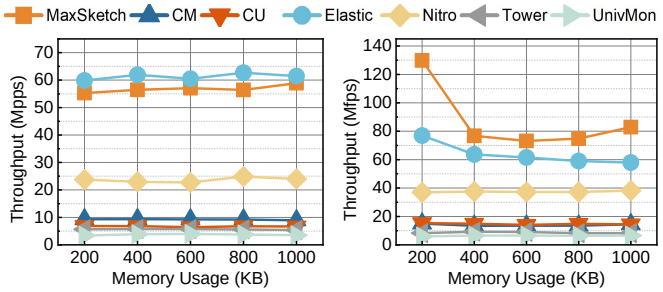
Experiment 3-6 (Heavy Hitter Detection of PolySketch). Figure 20 shows the precision, recall, and F1-score of the aggregation algorithm applied to the heavy hitter detection task. The aggregated sketch shows no overestimation, maintaining 100% precision across all configurations. In contrast, the baseline overestimates when multiple sketches are accumulated, causing fluctuations in precision.

Regarding recall, the baseline avoids additional underestimation errors, thus setting an upper limit on recall. Aggregating sketches from different slave nodes creates new hash collisions, resulting in extra underestimation errors. As the number of blocks increases, the recall gradually approaches the baseline, nearly matching it when using 5 blocks.

The F1-score reflects the aggregation algorithm's overall effectiveness in the heavy hitter detection task. With 5 blocks, the high precision and recall enable the F1-score to exceed the baseline in some instances.

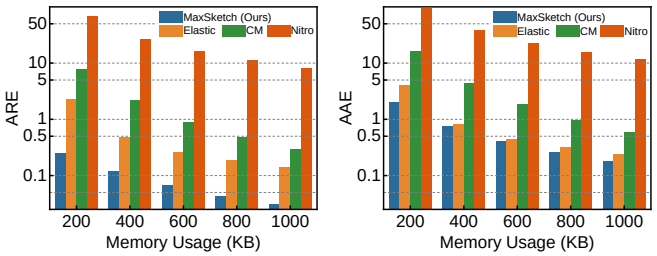
E. MaxSketch Performance Evaluation

In this part, we evaluate the performance of MaxSketch. Since the algorithms on PolySketch are based on MaxSketch, the performance of MaxSketch determines the upper limit of the system. First, we measure the throughput of each algorithm and use this metric to select candidates for subsequent accuracy evaluation. We then assess the performance of these selected algorithms in flow size estimation, heavy hitter detection, and entropy estimation tasks. We set the number of



(a) Insert (b) Query

Fig. 21: Throughput comparison for insert packets and query flow size estimation.



(a) ARE Comparison (b) AAE Comparison

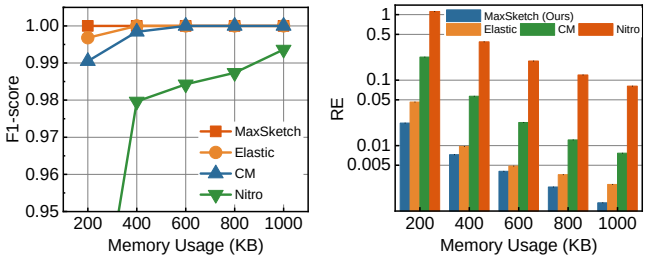
Fig. 22: Accuracy comparison for flow size estimation task.

rows to $d = 3$ for CM and CU, which is reported to have the best accuracy [11], and adjust the number of columns w to modify the memory usage of the sketch. Elastic Sketch and Nitrosketch use the same configuration as in their paper evaluation.

Experiment 4-1 (Throughput). Figure 21 compares the throughput of MaxSketch with several existing sketch structures, including CM-Sketch, CU-Sketch, Elastic Sketch, TowerSketch, Nitrosketch, and Univmon. Previous studies demonstrate that the majority of computational overhead in sketch insertion and query operations stems from hash calculations. By reducing the number of hash computations and adopting a streamlined data structure, MaxSketch achieves significantly higher throughput. However, MaxSketch remains a little lower than that of Elastic Sketch, which performs only one or fewer hash computations per packet. In query rates, because it also strongly dependents on hash, MaxSketch delivers substantially higher query rates than competing approaches.

To estimate the minimum packet processing rate required in data centers, we consider a typical 40 Gbps NIC environment. Assuming an average packet size of 900 bytes, based on the CAIDA dataset, a sketch must sustain at least 6 Mpps to satisfy baseline requirements. To provide redundancy, the safety threshold is typically set at twice this value, implying that a deployable sketch should support at least 12 Mpps. Algorithms that meet this threshold, specifically Elastic Sketch, Nitrosketch, and CM-Sketch, are included in the comparison.

Experiment 4-2 (Flow Size Estimation). Figure 22 compares the accuracy of MaxSketch, Elastic Sketch, CM-Sketch, and Nitrosketch in the flow size estimation task, demonstrating that MaxSketch significantly outperforms the competing algorithms. Notably, CM-sketch performs poorly at a memory usage of 200 KB. With a row setting of 3, they require additional memory to generate more columns, which helps avoid hash collisions. Nitrosketch sacrifices a considerable amount of accuracy in pursuit of high throughput, resulting in performance that is consistently and significantly lower than that of MaxSketch across various tasks. In comparison to the more complex multi-layered structure of Elastic Sketch, MaxSketch demonstrated superior performance while using less memory. With 200KB of memory, the ARE and AAE decreased by 88.6% and 50.7%, respectively. As memory usage increased to 1000KB, these values reduced by 72.9% and 25.9%, respectively.



(a) Heavy Hitter Detection (b) Entropy Estimation

Fig. 23: Accuracy comparison for heavy hitter detection and entropy estimation task.

Experiment 4-3 (Heavy Hitter Detection). Figure 23a compares the F1-scores of MaxSketch, Elastic Sketch, CM-Sketch, and Nitrosketch in the context of heavy-hitter detection. Elastic employs a dedicated data structure to monitor top-K flows, thereby maintaining high accuracy in the heavy hitter detection task. However, it still produces false detections when memory resources are constrained. In contrast, MaxSketch, which uses a unified data structure, achieves 100% accuracy across all experimental configurations. Compared with CM-Sketch and other sketches that also adopt unified data structures, MaxSketch exhibits a distinctly superior performance.

Experiment 4-4 (Entropy Estimation). Figure 23b compares the relative error of MaxSketch, Elastic Sketch, CM-Sketch, and Nitrosketch in the entropy estimation task, where the information entropy given by the formula $H = -\sum_k (k * \frac{n_k}{m} \log \frac{n_k}{m})$, where n_k is the number of size-k flows [18]. MaxSketch consistently outperforms all compared algorithms in the entropy estimation task. With memory allocations of 200 KB and 1000 KB, it achieves reductions in relative error of 49.2% and 48.5%, respectively, compared with Elastic Sketch.

VII. Conclusion

In this paper, we present HiMon, a lightweight distributed monitoring system specifically designed for data centers. HiMon employs a hierarchical sketch structure, consisting of MaxSketch and PolySketch, to monitor the traffic of each distributed node and aggregate the data to create a global traffic statistics view. Experimental results demonstrate that our approach significantly reduces CPU time and bandwidth consumption, while maintaining high accuracy across a range of tasks.

REFERENCES

[1] Anonymized Internet Traces 2019. https://catalog.caida.org/dataset/passive_2019_pcap. Dates used: January 17, 2019. Accessed: September 22, 2024.

[2] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (dctcp). *SIGCOMM Comput. Commun. Rev.*, 40(4):63–74, August 2010.

[3] Ran Ben Basat, Xiaoqi Chen, Gil Einziger, and Ori Rottenstreich. Designing heavy-hitter detection algorithms for programmable switches. *IEEE/ACM Transactions on Networking*, 28(3):1172–1185, 2020.

[4] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. Understanding data center traffic characteristics. *SIGCOMM Comput. Commun. Rev.*, 40(1):92–99, January 2010.

[5] Deyan Chen and Hong Zhao. Data security and privacy protection issues in cloud computing. In *2012 international conference on computer science and electronics engineering*, volume 1, pages 647–651. IEEE, 2012.

[6] Haoxian Chen, Nate Foster, Jake Silverman, Michael Whittaker, Brandon Zhang, and Rene Zhang. Felix: Implementing traffic measurement on end hosts using program analysis. In *Proceedings of the Symposium on SDN Research*, SOSR '16, New York, NY, USA, 2016. Association for Computing Machinery.

[7] Zhen Chen, Wenyu Dong, Hang Li, Peng Zhang, Xinming Chen, and Junwei Cao. Collaborative network security in multi-tenant data center for cloud computing. *Tsinghua Science and Technology*, 19(1):82–94, 2014.

[8] Cisco. netflow. http://www.cisco.com/.

[9] Andrew R. Curtis, Wonho Kim, and Praveen Yalagandula. Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection. In *2011 Proceedings IEEE INFOCOM*, pages 1629–1637, 2011.

[10] Anja Feldmann and Ward Whitt. Fitting mixtures of exponentials to long-tail distributions to analyze network performance models. *Performance Evaluation*, 31(3):245–279, 1998.

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

[12] Liyuan Gu, Ye Tian, Wei Chen, Zhongxiang Wei, Cenman Wang, and Xinming Zhang. Per-flow network measurement with distributed sketch. *IEEE/ACM Transactions on Networking*, 32(1):411–426, Jun 2023.

[13] Rob Harrison, Qizhe Cai, Arpit Gupta, and Jennifer Rexford. Network-wide heavy hitter detection with commodity switches. In *Proceedings of the Symposium on SDN Research*, pages 1–7, 2018.

[14] 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, 2023.

[15] InMon. sflow. http://www.sflow.org/.

[16] Diana Joumblatt, Renata Teixeira, Jaideep Chandrashekar, and Nina Taft. Hostview: annotating end-host performance measurements with user feedback. *SIGMETRICS Perform. Eval. Rev.*, 38(3):43–48, January 2011.

[17] Herald Kllapi, Eva Sitaridi, Manolis M Tsangaris, and Yannis Ioannidis. Schedule optimization for data processing flows on the cloud. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 289–300, 2011.

[18] 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.

[19] Fuliang Li, Kejun Guo, Jiaxing Shen, and Xingwei Wang. Effective network-wide traffic measurement: A lightweight distributed sketch deployment. In *IEEE INFOCOM 2024 - IEEE Conference on Computer Communications*, pages 181–190, 2024.

[20] Kate Ching-Ju Lin and Wei-Lun Lai. Mc-sketch: Enabling heterogeneous network monitoring resolutions with multi-class sketch. In *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*. IEEE, May 2022.

[21] Masoud Moshref, Minlan Yu, Ramesh Govindan, and Amin Vahdat. Trumpet: Timely and precise triggers in data centers. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, page 129–143, New York, NY, USA, 2016. Association for Computing Machinery.

[22] Konstantin Pilz and Lennart Heim. Compute at scale: A broad investigation into the data center industry. *arXiv preprint arXiv:2311.02651*, 2023.

[23] Long Qu, Chadi Assi, and Khaled Shaban. Delay-aware scheduling and resource optimization with network function virtualization. *IEEE Transactions on communications*, 64(9):3746–3758, 2016.

[24] Junaid Shuja, Sajjad A Madani, Kashif Bilal, Khizar Hayat, Samee U Khan, and Shahzad Sarwar. Energy-efficient data centers. *Computing*, 94(12):973–994, 2012.

[25] Vibhaalakshmi Sivaraman, Srinivas Narayana, Ori Rottenstreich, S. Muthukrishnan, and Jennifer Rexford. Heavy-hitter detection entirely in the data plane. In *Proceedings of the Symposium on SDN Research*, SOSR '17, page 164–176, New York, NY, USA, 2017. Association for Computing Machinery.

[26] Vibhaalakshmi Sivaraman, Srinivas Narayana, Ori Rottenstreich, Shan Muthukrishnan, and Jennifer Rexford. Heavy-hitter detection entirely in the data plane. In *Proceedings of the Symposium on SDN Research*, pages 164–176, 2017.

[27] Cha Hwan Song, Pravein Govindan Kannan, Bryan Kian Hsiang Low, and Mun Choon Chan. Fcm-sketch: generic network measurements with data plane support. In *Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies*, pages 78–92, 2020.

[28] Praveen Tammana, Rachit Agarwal, and Myungjin Lee. Simplifying datacenter network debugging with PathDump. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 233–248, Savannah, GA, November 2016. USENIX Association.

[29] Praveen Tammana, Rachit Agarwal, and Myungjin Lee. Distributed network monitoring and debugging with SwitchPointer. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 453–456, Renton, WA, April 2018. USENIX Association.

[30] Jin Wang, Shuying Rao, Ying Liu, Pradip Kumar Sharma, and Jinbin Hu. Load balancing for heterogeneous traffic in datacenter networks. *Journal of Network and Computer Applications*, 217:103692, 2023.

[31] 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.

[32] Xiwen Yu, Hongli Xu, Da Yao, Haibo Wang, and Liusheng Huang. Countmax: A lightweight and cooperative sketch measurement for software-defined networks. *IEEE/ACM Transactions on Networking*, 26(6):2774–2786, 2018.

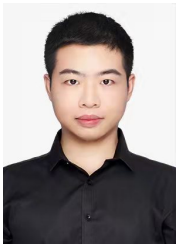
[33] Yin Zhang, Sumeet Singh, Subhabrata Sen, Nick Duffield, and Carsten Lund. Online identification of hierarchical heavy hitters: algorithms, evaluation, and applications. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, pages 101–114, 2004.

Yuchen Zhang (Student Member, IEEE) received B.Eng from Zhejiang University in 2021. He is currently pursuing a Ph.D. in the College of Control Science and Engineering, Zhejiang University, Hangzhou, China. His research interests include cloud networking and load balancing.

Zhenyu Wen (Senior Member, IEEE) is currently a Tenure-tracked professor with the Institute of Cyberspace Security and College of Information Engineering, Zhejiang University of Technology, and is a postdoctoral researcher with the University of Science and Technology of China. His current research interests include IoT, crowd sources, AI systems, and cloud computing. For his contributions to the area of scalable data management for the Internet of Things. He was awarded the IEEE TCSC Award for Excellence in Scalable Computing (Early Career Researchers) in 2020.

Shibo He (Senior Member, IEEE) received the Ph.D. degree in control science and engineering from Zhejiang University, Hangzhou, China, in 2012. From November 2010 to November 2011, he was a Visiting Scholar with the University of Waterloo, Waterloo, ON, Canada. From March 2014 to May 2014, he was an Associate Research Scientist and from May 2012 to February 2014, a Postdoctoral Scholar with Arizona State University, Tempe, AZ, USA. He is currently a Professor with Zhejiang University. His research interests include Internet of Things, crowdsensing, and Big Data analysis. Prof. He is on the editorial board of the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, Peer-to-Peer Networking and Applications, and KSII Transactions on Internet and Information Systems. He is also the Guest Editor of Computer Communications and International Journal of Distributed Sensor Networks. He was a Symposium Co-Chair of the IEEE GlobeCom 2020 and IEEE ICC 2017, a TPC Co-Chair of the i-Span 2018, a Finance and Registration Chair of the ACM MobiHoc 2015, a TPC Co-Chair of the IEEE ScalCom 2014, a TPC Vice Co-Chair of the ANT 2013'lC2014, a Track Co-Chair of the Pervasive Algorithms, Protocols, and Networks of EUSPN 2013, a Web Co-Chair of the IEEE MASS 2013, and a Publicity Co-Chair of the IEEE WiSARN 2010 and FCN 2014.

Xiang Chen (Member, IEEE) received the Ph.D. degree from the College of Computer Science and Technology, Zhejiang University, China, in 2026. He is currently a Research Fellow with the College of Computer Science and Technology, Zhejiang University. Xiang has received the Best Paper Award from IEEE/ACM IWQoS 2021, and the Best Paper Candidate from IEEE INFOCOM 2021. He has published papers in top-tier conferences such as ACM SIGCOMM, USENIX NSDI, and ACM EuroSys, and journals such as IEEE COMST and TON. His current research interests focus on network measurements.



Chaojie Gu (Member, IEEE) received the B.Eng. degree in information security from the Harbin Institute of Technology, Weihai, China, in 2016, and the Ph.D. degree in computer science and engineering from Nanyang Technological University, Singapore, in 2020. He was a Research Fellow with Singtel Cognitive and Artificial Intelligence Lab for Enterprise in 2021. He is currently an Assistant Professor with the College of Control Science and Engineering, Zhejiang University, Hangzhou, China. His research interests include IoT, industrial IoT, edge computing, and low-power wide area networks.

Jiming Chen (Fellow, IEEE) received B.Sc and PhD degree both in Control Science and Engineering from Zhejiang University, Hangzhou, China in 2000 and 2005, respectively. He is currently a professor with the Department of Control Science and Engineering at Zhejiang University and president of Hangzhou Dianzi University. His research interests include IoT, networked control, wireless networks. He serves on the editorial boards of multiple IEEE Transactions, and the general co-chairs for IEEE RTCSA'19, IEEE Datacom'19 and IEEE PST'20. He was a recipient of the 7th IEEE ComSoc Asia/Pacific Outstanding Paper Award, the JSPS Invitation Fellowship, and the IEEE ComSoc AP Outstanding Young Researcher Award. He is an IEEE VTS distinguished lecturer. He is a fellow of the CAA.

APPENDIX

In this section, we first analyze the time and space complexity of MaxSketch. Next, we provide an error analysis for querying the preferred flow within each bucket of MaxSketch, demonstrating the superiority of our update algorithm.

A. Space and Time Complexities

Theorem 1. The space usage is $O(d*w)$. The update time is $O(d)$ per packet, and the query time is $O(d)$ per flow.

Proof. Each bucket of MS-sketch employs a fixed amount of memory for counting. So, the space usage of MS-sketch is $O(d*w)$. The update and query operations need to access d buckets and perform $2*d$ hash calculations. Therefore, the update and query take $O(d)$ time.

B. Error Bound of MaxSketch

Theorem 2. The estimate in PC is always less than or equal to the accurate value, unless two flows collide both on the bucket and the ID hash at the same time.

Proof. If there is no flow collision, the estimated value is equal to the actual value. If there is a flow collision, we provide the proof in Theorem 3.

Meanwhile, we give the probability that the flows collide in both the bucket and the ID hash. For a hash computation with a hash space of size N and i flows, the probability that no collisions occur among any pair of flows is

$$P = \frac{A_N^i}{N^i} \tag{1}$$

We illustrate this probability using a 200 KB MaxSketch as an example. This configuration includes 14,628 buckets, each storing a flow ID in 2 bytes, corresponding to a hash space of 65,535. The CAIDA dataset employed in the experiments contains an average of 73,920 flows. Assuming a uniform hash distribution, each bucket stores approximately five flows after the first round of hashing.

For a single bucket, the probability of an ID collision among five flows is

$$P = 1 - \frac{A_{65535}^5}{65535^5} \approx 1.5258 \times 10^{-4} \tag{2}$$

Thus, the expected number of hash collisions in the entire MaxSketch is

$$14628 \times 1.5258 \times 10^{-4} \approx 2.2320 \tag{3}$$

In evaluation, the observed number of collisions ranges from 1.4 to 3.3, depending on the hash function used. These results are consistent with the theoretical estimate.

Theorem 3. When a large flow f_1 comes into conflict with a small flow f_2, if n packets of f_2 emerge before the n-th packet of f_1, it will inevitably result in an error in the statistics of flow f_1, which is $n-1$ packets less than the actual value.

Proof. Take any instance where flow f_1 is exchanged from NC to PC as an example. Suppose at this moment the count value of PC is m and that of NC is $m+1$, which triggers

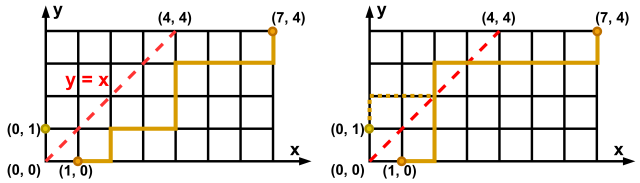


Fig. 24: (a) Legal path; (b) Mapping between illegal paths.

the exchange of counters. PC saves the estimate of flow f_1 and initializes it to 2; NC saves the estimate of flow f_2 and initializes it to 1. Both flows lose the estimate of $m-1$ packets. Therefore, each packet of f_2 will result in a one-count estimation bias of f_1. And as all n packets of flow f_2 occur before the n-th packet of flow f_1, the estimation error of f_2 is equal to f_1. Since the final count value of NC is 1, flow f_2 causes an estimated error of $n-1$ packets for flow f_1.

Lemma 1. If there are α_1 packets in flow f_1 and α_2 packets in flow f_2 $(\alpha_1 \geq \alpha_2)$, then the probability that all α_2 packets of flow f_2 appear before the α_2-th packet of flow f_1 is

$$\mathbb{P} = \frac{C_{2y-1}^{\alpha_2}}{C_{\alpha_2+\alpha_1}^{\alpha_2}} = \frac{(2y-1)!\alpha_1!}{(\alpha_2+\alpha_1)!(\alpha_2-1)!} \tag{4}$$

Proof. Randomly insert α_2 packets of flow f_2 into flow f_1 (place α_2 balls in α_1+1 buckets). There are a total of $C_{\alpha_2+\alpha_1}^{\alpha_2}$ different insertion methods. Among them, there are a total of $C_{2y-1}^{\alpha_2}$ schemes that guarantee that all packets of flow f_2 are inserted before the α_2-th packet of flow f_1 (put α_2 balls in α_2 buckets). Therefore, $\mathbb{P} = C_{2y-1}^{\alpha_2}/C_{\alpha_2+\alpha_1}^{\alpha_2}$.

Lemma 2. If there are α_1 packets in flow f_1 and α_2 packets in flow f_2 $(\alpha_1 \geq \alpha_2)$, the probability that the i-th packet $(i = 1, \cdots, \alpha_2)$ of flow f_2 occurs after the i-th packet of flow f_1 is

$$\mathbb{P} = 1 - \frac{\alpha_2}{\alpha_1+1} \tag{5}$$

Proof. There are α_1 packets in flow f_1, thus, there are α_1+1 positions where α_2 packets of flow f_2 can be inserted. Therefore, the problem can be abstracted as repeatedly placing α_2 identical balls in α_1+1 buckets. The first ball cannot be placed in the first bucket. The i-th ball must be placed after the $(i-1)$-th ball and the i-th bucket. $(i = 2, \cdots, \alpha_2)$.

We employ a graphical approach to explain this formula. Let $\alpha_1 + 1 = \lambda_1$ and $\alpha_2 = \lambda_2$. Now we construct a grid of λ_2 rows and λ_1 columns. As depicted in Figure 24, taking $\lambda_1 = 7$ and $\lambda_2 = 4$ as an example. All placement schemes that fulfill the conditions correspond one-to-one to a path starting from $(1, 0)$ and reaching (λ_1, λ_2) and not passing through the dashed line $y = x$.

The number of paths from $(1, 0)$ to (λ_1, λ_2) is $C_{\lambda_1+\lambda_2-1}^{\lambda_1-1}$. Taking the symmetric point (0, 1) of (1, 0) with respect to $y = x$, we find that every path starting from (0, 1) and reaching (λ_1, λ_2) will pass through $y = x$ and corresponds to a unique path starting from (1, 0) and reaching (λ_1, λ_2) and passing through $y = x$. Therefore, the number of paths fulfilling the conditions is equal to the total number of paths minus the number of paths passing through $y = x$, which is

$$Path = C_{\lambda_1+\lambda_2-1}^{\lambda_1-1} - C_{\lambda_1+\lambda_2-1}^{\lambda_1} = \frac{\lambda_1-\lambda_2}{\lambda_1} C_{\lambda_1+\lambda_2-1}^{\lambda_1-1} \tag{6}$$

$$\mathbb{P} = \frac{\frac{\lambda_1-\lambda_2}{a} C_{\lambda_1+\lambda_2-1}^{\lambda_1-1}}{C_{\lambda_1+\lambda_2-1}^{\lambda_1-1}} = \frac{\lambda_1-\lambda_2}{\lambda_{\ 1}} = 1 - \frac{\alpha_2}{\alpha_1+1} \tag{7}$$

Theorem 4. There are α_1 packets in flow f_1 and α_2 packets in flow f_2 $(\alpha_1 \geq \alpha_2)$. Find the largest β such that the first β packets of flow f_2 all appear before the β-th packet of flow f_1.

$$\mathbb{P}\{X = \beta\} = \frac{(\alpha_1-\alpha_2+1) * C_{2\beta-1}^{\beta} * C_{\alpha_1+\alpha_2-2\beta}^{\alpha_1-\beta}}{(\alpha_1-\beta+1) * C_{\alpha_1+\alpha_2}^{\alpha_2}} \tag{8}$$

Proof. The total number of combinations of α_1 packets and α_2 packets is $C_{\alpha_1+\alpha_2}^{\alpha_2}$. The packets that fulfill the condition comprise two independent parts: (1) The number of combinations where the first β packets of flow f_2 occur before the β-th packet of flow f_1 is $C_{2\beta-1}^{\beta}$; (2) For the latter $\alpha_2-\beta$ packets of flow f_2, the probability that the i-th packet $(i = \beta+1, \cdots, \alpha_2)$ does not appear before the i-th packet of flow f_1 is provided by Lemma 3. Therefore, the total number of combinations is $(1-\frac{\alpha_2-\beta}{\alpha_1-\beta+1})C_{\alpha_1+\alpha_2-2\beta}^{\alpha_1-\beta}$. Since the two parts are independent, thus $\mathbb{P}\{X = z\} = C_{2\beta-1}^{\beta} * (1 - \frac{\alpha_2-\beta}{\alpha_1-\beta+1}) * C_{\alpha_1+\alpha_2-2\beta}^{\alpha_1-\beta}/C_{\alpha_1+\alpha_2}^{\alpha_2}$, which is equivalent to formula (8).

Theorem 5. When large flow f_1 conflicts with small flow f_2, the expected statistical error of flow f_1 is

$$\mathbb{E}(X) = \sum_{i=1}^{\alpha_2} (i-1) * \mathbb{P}\{X = i\} \tag{9}$$

Proof. According to Theorem 4, when i packets of flow f_2 occur before the i-th packet of flow f_1, it will lead to an error of $i-1$ packets in the statistics of flow f_1. According to Theorem 5, this probability is $\mathbb{P}\{X = i\}$. Hence, $\mathbb{E}(X) = \sum_{i=1}^{\alpha_2}(i-1) * \mathbb{P}\{X = i\}$.

When $\alpha_2 > 1$ and $\alpha_1 \gg \alpha_2$, $\mathbb{E}(X) \approx \alpha_2/\alpha_1$. For example, when $\alpha_2 = 5$ and $\alpha_1 = 200$, $\mathbb{E}(X) = 0.0264$; when $\alpha_2 = 20$ and $\alpha_1 = 2000$, $\mathbb{E}(X) = 0.0103$.