MV-Sketch: A Fast and Compact Invertible Sketch for Heavy Flow Detection in Network Data Streams

Lu Tang[1], Qun Huang[2], and Patrick P. C. Lee[1]
[1]Department of Computer Science and Engineering, The Chinese University of Hong Kong
[2]State Key Lab of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences

***Abstract*—Fast detection of heavy flows (e.g., heavy hitters and heavy changers) in massive network traffic is challenging due to the stringent requirements of fast packet processing and limited resource availability. Invertible sketches are summary data structures that can recover heavy flows with small memory footprints and bounded errors, yet existing invertible sketches incur high memory access overhead that leads to performance degradation. We present MV-Sketch, a fast and compact invertible sketch that supports heavy flow detection with small and static memory allocation. MV-Sketch tracks candidate heavy flows inside the sketch data structure via the idea of majority voting, such that it incurs small memory access overhead in both update and query operations, while achieving high detection accuracy. We present theoretical analysis on the memory usage, performance, and accuracy of MV-Sketch. Trace-driven evaluation shows that MV-Sketch achieves higher accuracy than existing invertible sketches, with up to 3.38× throughput gain. We also show how to boost the performance of MV-Sketch with SIMD instructions.**

I. Introduction

Identifying abnormal patterns of *flows* (e.g., hosts, source-destination pairs, or 5-tuples) in massive network traffic is essential for various network management tasks, such as traffic engineering, load balancing, and intrusion detection. Two types of abnormal flows are of particular interest: *heavy hitters* (i.e., flows that generate an unexpectedly high volume of traffic) and *heavy changers* (i.e., flows that generate an unexpectedly high change of traffic volume in a short duration). By identifying both heavy hitters and heavy changers (collectively referred to as *heavy flows*), network operators can quickly respond to performance outliers, mis-behaved usage, and potential DDoS attacks, so as to maintain network stability and QoS guarantees.

Unfortunately, the stringent requirements of fast packet processing and limited memory availability pose challenges to practical heavy flow detection. First, the packet processing rate of heavy flow detection must keep pace with the ever-increasing network speed, especially in the worst case when traffic bursts or attacks happen [17]. For example, a fully utilized 10Gb/s link with a minimum packet size of 64 bytes implies that the heavy flow detection algorithm must process at least 14.88M packets per second. In addition, the available memory footprints are constrained in practice. While per-flow monitoring with linear hash tables is arguably feasible in software [1], it requires tremendous memory space in the worst case. For example, monitoring all 5-tuple flows requires to track a maximum of 2^{104} flow entries.

Given the rigid packet processing and memory requirements, many approaches perform approximate heavy flow detection via *sketches*, which are summary data structures that significantly mitigate memory footprints with bounded detection errors. Classical sketches [10], [11], [22] are proven effective, yet they are *non-invertible*: while we can query a sketch whether a specific flow is a heavy flow, we cannot readily recover all heavy flows from only the sketch data structure itself. Instead, we must query every possible flow to check whether it is a heavy flow. Such a brute-force approach is computationally expensive for an extremely large flow key space (e.g., the size is 2^{104} for 5-tuple flows).

This motivates us to explore *invertible sketches*, which provide provable error bounds as in classical sketches, while supporting the queries of recovering all heavy flows. Invertible sketches are well studied in the literature (e.g., [8], [11], [13], [18], [24], [29]) for heavy flow detection. However, there remain limitations in existing invertible sketches. In particular, they either maintain heavy flows in external DRAM-based data structures [11], [18], or track flow keys in smaller-size bits or sub-keys [8], [13], [24], [29]. We argue that both approaches incur substantial memory access overhead that leads to degraded processing performance (see Section II-B).

In this paper, we present MV-Sketch, a fast and compact invertible sketch for heavy flow detection. It tracks candidate heavy flow keys together with the counters in a sketch data structure, and updates the candidate heavy flow keys based on the *majority vote algorithm* [7] in an online streaming fashion. A key design feature of MV-Sketch is that it maintains a sketch data structure with *small* and *static* memory allocation (i.e., no dynamic memory allocation is needed). This allows lightweight memory access in both update and detection operations, and provides viable opportunities for hardware acceleration. To summarize, we make the following contributions.

- We design MV-Sketch, an invertible sketch that supports both heavy hitter and heavy changer detection and can be generalized for distributed detection. See Section III.
- We present theoretical analysis on MV-Sketch for its memory space complexity, update/detection time complexity, and detection accuracy. See Section IV.
- We show via trace-driven evaluation that MV-Sketch achieves higher detection accuracy for most memory configurations and up to 3.38× throughput gain compared to state-of-the-art invertible sketches. Furthermore, we extend MV-Sketch with *Single Instruction, Multiple Data (SIMD)* instructions to boost its update performance. See Section V.

The source code of our MV-Sketch prototype is available at: **http://adslab.cse.cuhk.edu.hk/software/mvsketch**.

II. Background

A. Heavy Flow Detection

We consider a stream of packets, each of which is denoted by a key-value pair (x, v_x), where x is a key drawn from a domain $[n] = \{0, 1, \cdots, n-1\}$ and v_x is the value of x. In network measurement, x is the flow identifier (e.g., source/destination address pairs or 5-tuples), while v_x is either one (for packet counting) or the packet size (for byte counting). We conduct measurement at regular time intervals called *epochs*.

We formally define heavy hitters and heavy changers as follows. Let ϕ be a pre-defined fractional threshold (where $0 < \phi < 1$) that is used to differentiate heavy flows from network traffic (we use the same ϕ for both heavy hitter and heavy changer detection for simplicity). Let $S(x)$ be the sum (of all v_x's) of flow x in an epoch, and $D(x)$ be the absolute change of $S(x)$ of flow x across two epochs. Let $\mathcal{S}$ be the total sum of all flows in an epoch (i.e., $\mathcal{S} = \sum_{x \in [n]} S(x)$), and $\mathcal{D}$ be the total absolute change of all flows across two epochs (i.e., $\mathcal{D} = \sum_{x \in [n]} D(x)$). Both $\mathcal{S}$ and $\mathcal{D}$ can be obtained in practice: for $\mathcal{S}$, we can maintain an extra counter that counts the total traffic; for $\mathcal{D}$, we can run an l_1-streaming algorithm and estimate $\mathcal{D}$ (equivalent to the l_1-distance) in one pass [28]. Finally, flow x is said to be a *heavy hitter* if $S(x) \geq \phi\mathcal{S}$, or a *heavy changer* if $D(x) \geq \phi\mathcal{D}$.

B. Sketches

Sketches are summary data structures that track values in a fixed number of entries called *buckets*. Classical sketches on heavy flow detection (e.g., Count Sketch [10], K-ary Sketch [22], and Count-Min Sketch [11]) represent a sketch as a two-dimensional array of buckets and provide different theoretical trade-offs across memory usage, performance, and accuracy.

Take Count-Min Sketch [11] as an example. We construct the sketch as r rows of buckets, with w buckets in each row. Each bucket is associated with a counter initialized as zero. For each tuple (x, v_x) received in an epoch, we hash x into a bucket in each of the r rows using r pairwise independent hash functions. We increment the counter in each of the r hashed buckets by v_x. Since multiple flows can be hashed to the same bucket, we can only provide an estimate for the sum of a flow. Count-Min Sketch uses the minimum counter value of all r hashed buckets as the estimated sum of a flow. We can check if a flow is a heavy hitter by checking if its estimated sum exceeds the threshold; similarly, we can check if a flow is a heavy changer by checking if the absolute change of its estimated sums in two epochs exceeds the threshold. However, Count-Min Sketch is non-invertible, as we must check every flow in the entire flow key space to recover all heavy flows; note that Count Sketch and K-ary Sketch are also non-invertible.

Invertible sketches (e.g., [8], [11], [13], [18], [24], [29]) allow all heavy flows to be recovered from only the sketch data structure itself. State-of-the-art invertible sketches can be classified into three types.

Extra data structures. Count-Min-Heap [11] is an augmented Count-Min Sketch that uses a heap to track all candidate heavy flows and their estimated sums. If any incoming flow whose estimated sum exceeds the threshold, it is added to the heap. LD-Sketch [18] maintains a two-dimensional array of buckets and links each bucket with an associative array to track the candidate heavy flows that are hashed to the bucket. However, updating a heap or an associative array incurs high memory access overhead, which increases with the number of heavy flows. In particular, LD-Sketch occasionally expands the associative array to hold more candidate heavy flows, yet dynamic memory allocation is a costly operation and difficult to implement in hardware [3].

Group testing. Deltoid [13] comprises multiple counter groups with $1 + L$ counters each (where L is the number of bits in a key), in which one counter tracks the total sum of the group, and the remaining L counters correspond to the bit positions of a key. It maps each flow key to a subset of groups and increments the counters whose corresponding bits of the key are one. To recover heavy flows, Deltoid first identifies all groups whose total sums exceed the threshold. If each such group has only one heavy flow, the heavy flow can be recovered: the bit is one if a counter exceeds the threshold, or zero otherwise. Fast Sketch [24] is similar to Deltoid except that it maps the quotient of a flow key to the sketch. However, both Deltoid and Fast Sketch have high update overhead, as their numbers of counters increase with the key length.

Enumeration. Reversible Sketch [29] finds heavy flows by pruning the enumeration space of flow keys. It divides a flow key into smaller sub-keys that are hashed independently, and concatenates the hash results to identify the hashed buckets. To recover heavy flows, it enumerates each sub-key space and combines the recovered sub-keys to form the heavy flows. SeqHash [8] follows a similar design, yet it hashes the key prefixes of different lengths into multiple smaller sketches. However, the update costs of both Reversible Sketch and SeqHash increase with the key length.

III. MV-Sketch Design

MV-Sketch is a novel invertible sketch for heavy flow detection and aims for the following design goals:

- **Invertibility:** MV-Sketch is invertible and readily returns all heavy flows (i.e., heavy hitters or heavy changers) from only the sketch data structure itself.
- **High detection accuracy:** MV-Sketch supports accurate heavy flow detection with provable error bounds.
- **Small and static memory:** MV-Sketch maintains compact data structures with small memory footprints. Also, it can be constructed with static memory allocation, which mitigates memory management overhead as opposed to dynamic memory allocation in LD-Sketch [18].
- **High processing speed:** MV-Sketch processes packets at high speed by limiting the memory access overhead of per-packet updates. It also takes advantage of static memory allocation to allow hardware acceleration.

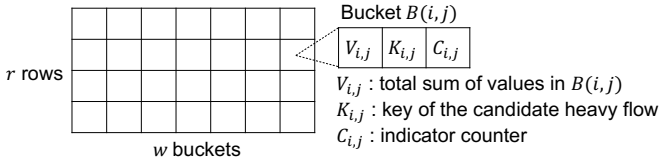


Fig. 1. Data structure of MV-Sketch.

- **Distributed detection:** MV-Sketch can be extended for distributed detection with multiple MV-Sketch instances.

A. Main Idea

Like Count-Min Sketch [11], MV-Sketch is initialized as a two-dimensional array of buckets (see Section II-B), in which each bucket tracks the values of the flows that are hashed to itself. MV-Sketch augments Count-Min Sketch in a way that each bucket also tracks a *candidate heavy flow* that has a high likelihood of carrying the largest amount of traffic among all flows that are hashed to the bucket. Our rationale is that in practice, a small number of large flows dominate in IP traffic [33]. Thus, the candidate heavy flow is very likely to carry much more traffic than all other flows that are hashed to the same bucket. Also, by using a sufficient number of buckets, we can significantly reduce the probability that two heavy flows are hashed into the same bucket, and hence accurately track multiple heavy flows.

To find the candidate heavy flow in each bucket, we apply the *majority vote algorithm (MJRTY)* [7], which enables us to track the candidate heavy flow in an online streaming fashion. MJRTY processes a stream of votes (corresponding to packets in our case), each of which has a vote key and a vote count one. It aims to find the *majority vote*, defined as the vote key that has more than half of the total vote counts, from the stream of votes in one pass with constant memory usage. At any time, it stores (i) the *candidate majority vote* that is thus far observed in a stream and (ii) an *indicator counter* that tracks whether the currently stored vote remains the candidate majority vote. Initially, it stores the first vote and initializes the indicator counter as one. Each time when a new vote arrives, MJRTY compares the new vote with the candidate majority vote. If both votes are the same (i.e., the same vote key), it increments the indicator counter by one; otherwise, it decrements the indicator counter by one. If the indicator counter is below zero, MJRTY replaces the current candidate majority vote with the new vote and resets the counter to one. MJRTY ensures that the true majority vote must be the candidate majority vote stored by MJRTY at the end of the stream [7].

B. Data Structure of MV-Sketch

Figure 1 shows the data structure of MV-Sketch, which is composed of a two-dimensional array of buckets with r rows and w columns. Let $B(i,j)$ denote the bucket at the i-th row and the j-th column, where $1 \le i \le r$ and $1 \le j \le w$. Each bucket $B(i,j)$ consists of three fields: (i) $V_{i,j}$, which counts the total sum of values of all flows hashed to the bucket; (ii) $K_{i,j}$, which tracks the key of the current candidate heavy flow in the bucket; and (iii) $C_{i,j}$, which is the indicator counter that checks if the candidate heavy flow should be kept or replaced as in MJRTY [7]. In addition, MV-Sketch is associated with r pairwise-independent hash functions, denoted by $h_1 \dots h_r$, such that each h_i (where $1 \le i \le r$) hashes the key $x \in [n]$ of each incoming packet to one of the w buckets in row i. Note that the data structure has a fixed memory size and can be pre-allocated in advance.

Algorithm 1 Update

Input:(x, v_x)
1: **for** $i = 1$ to r **do**
2: $V_{i,h_i(x)} \leftarrow V_{i,h_i(x)} + v_x$
3: **if** $K_{i,h_i(x)} = x$ **then**
4: $C_{i,h_i(x)} \leftarrow C_{i,h_i(x)} + v_x$
5: **else**
6: $C_{i,h_i(x)} \leftarrow C_{i,h_i(x)} - v_x$
7: **if** $C_{i,h_i(x)} < 0$ **then**
8: $K_{i,h_i(x)} \leftarrow x$
9: $C_{i,h_i(x)} \leftarrow -C_{i,h_i(x)}$
10: **end if**
11: **end if**
12: **end for**

C. Basic Operations

MV-Sketch supports two basic operations: (i) *Update*, which inserts each incoming packet into the sketch; (ii) *Query*, which returns the estimated sum of a given flow in an epoch.

Algorithm 1 shows the Update operation. All fields $V_{i,j}$, $K_{i,j}$, and $C_{i,j}$ are initialized as zero for $B(i,j)$, where $1 \le i \le r$ and $1 \le j \le w$. For each (x, v_x), we hash x into $B(i,j)$ in the i-th row with $j = h_i(x)$ for $1 \le i \le r$. We first increment $V_{i,j}$ by v_x (Line 2). We then check if x is stored in $K_{i,j}$ based on the MJRTY algorithm: if $K_{i,j}$ equals x, we increment $C_{i,j}$ by v_x (Lines 3-4). Otherwise, we decrement $C_{i,j}$ by v_x (Lines 5-6); if $C_{i,j}$ drops below zero, we replace $K_{i,j}$ by x and reset $C_{i,j}$ with its absolute value (Lines 7-10). Note that the Update operation differs from MJRTY as it supports general value counts (or the number of bytes) with any non-negative value v_x, while MJRTY considers only vote counts (or the number of packets) with v_x always being one.

Algorithm 2 shows the Query operation. For each hashed bucket in row i (where $1 \le i \le r$), we calculate a row estimate $\hat{S}_i(x)$ of flow x (Lines 1-7): if x and $K_{i,j}$ are the same, we set $\hat{S}_i(x) = (V_{i,j} + C_{i,j})/2$; otherwise, we set $\hat{S}_i(x) = (V_{i,j} - C_{i,j})/2$. Finally, we return the final estimate $\hat{S}(x)$ as the minimum of all row estimates (Lines 8-9).

D. Heavy Flow Detection

Heavy hitter detection. To detect heavy hitters, we check every bucket $B(i,j)$ ($1 \le i \le r$ and $1 \le j \le w$) at the end of an epoch. For each $B(i,j)$, if $V_{i,j} \ge \phi\mathcal{S}$, we let $x = K_{i,j}$ and query $\hat{S}(x)$ from Algorithm 2; if $\hat{S}(x) \ge \phi\mathcal{S}$, we report x as a heavy hitter.

Heavy changer detection. To detect heavy changers, we compare two sketches at the ends of two epochs. One possible detection approach is to exploit the linear property of sketches as in prior studies [13], [24], [29], in which we compute

Algorithm 2 Query

Input: flow key x
Output: estimate $\hat{S}(x)$ of flow x

```
for i = 1 to r do
    if K_{i,h_i(x)} = x then
        Ŝ_i(x) ← (V_{i,h_i(x)} + C_{i,h_i(x)})/2
    else
        Ŝ_i(x) ← (V_{i,h_i(x)} − C_{i,h_i(x)})/2
    end if
end for
Ŝ(x) ← min_{1≤i≤r}{Ŝ_i(x)}
return Ŝ(x)
```

the differences of $V_{i,j}$'s of the buckets at the same positions across the two sketches and recover the flows from the buckets whose differences exceed the threshold $\phi\mathcal{D}$ (see Section II-A). However, such an approach can return many false negatives, since the hash collisions of two heavy changers, one with a high incremental change and another with a high decremental change, can cancel out the changes of each other.

To reduce the number of false negatives, we instead use the *estimated maximum change* of a flow for heavy changer detection. Specifically, let $U(x)$ and $L(x)$ be the upper and lower bounds of $S(x)$, respectively. We set $U(x) = \hat{S}(x)$ returned by Algorithm 2. Also, we set $L(x) = \max_{1\le i\le r}\{L_i(x)\}$, where $L_i(x)$ is set as follows: for each hashed bucket $B(i,j)$ of x (where $1 \le i \le r$ and $j = h_i(x)$), if $K_{i,j}$ equals x, we set $L_i(x) = C_{i,j}$; otherwise, we set $L_i(x) = 0$. Note that both $U(x)$ and $L(x)$ are the *true* upper and lower bounds of $S(x)$, respectively (see Lemma 2 in Section IV-B). Now, let $U^1(x)$ and $L^1(x)$ (resp. $U^2(x)$ and $L^2(x)$) be the upper and lower bounds of $S(x)$ in the previous (resp. current) epoch, respectively. Then the estimated maximum change of flow x is given by $\hat{D}(x) = \max\{|U^1(x) - L^2(x)|, |L^1(x) - U^2(x)|\}$.

We now detect heavy changers as follows. We check every bucket $B(i,j)$, ($1 \le i \le r$ and $1 \le j \le w$) of two sketches of the previous and current epochs. For each $B(i,j)$ in each of the sketches, if $V_{i,j} \ge \phi\mathcal{D}$, we let $x = K_{i,j}$ and estimate $\hat{D}(x)$; if $\hat{D}(x) \ge \phi\mathcal{D}$, we report x as a heavy changer.

Currently, MV-Sketch is designed to detect heavy hitters and heavy changers, both of which focus on the values (e.g., packet or byte counts) of a flow. We can extend MV-Sketch to monitor hosts with a high number of distinct connections in DDoS or superspreader detection by either associating the buckets with approximate distinct counters [12] or filtering duplicate connections with a Bloom filter [34]. We pose the analysis of DDoS and superspreader detection as future work.

E. Distributed Heavy Flow Detection

We extend MV-Sketch for distributed heavy flow detection based on the distributed architecture [18]. We deploy $q \ge 1$ detectors, each of which deploys an MV-Sketch to monitor packets from multiple streaming sources. Suppose that each streaming source maps a flow to a subset d out of q detectors, where $d \le q$, and distributes each packet of the flow uniformly to one of the d selected detectors. At the end of each epoch, each detector sends the local detection results to a centralized controller for final heavy flow detection.

For heavy hitter detection, each detector checks every bucket $B(i,j)$ in MV-Sketch. Let $x = K_{i,j}$, and if $\hat{S}(x) \ge \frac{\phi}{d}\mathcal{S}$, the detector sends the tuple $(x, \hat{S}(x))$ of flow x to the controller. After collecting all results from q detectors, the controller adds the estimates of each flow. If the added estimate of a flow exceeds $\phi\mathcal{S}$, the flow is reported as a heavy hitter.

For heavy changer detection, each detector checks every bucket $B(i,j)$ of two sketches of the previous and current epochs. If $V_{i,j} \ge \frac{\phi}{d}\mathcal{D}$, it lets $x = K_{i,j}$ and estimates $\hat{D}(x)$; if $\hat{D}(x) \ge \frac{\phi}{d}\mathcal{D}$, the detector sends the tuple $(x, \hat{D}(x))$ of flow x to the controller. Similar to heavy hitter detection, the controller adds the estimates of each flow from q detectors. If the added estimate of a flow exceeds $\phi\mathcal{D}$, it is reported as a heavy changer.

IV. Theoretical Analysis

We present theoretical analysis on MV-Sketch in heavy flow detection. We also compare MV-Sketch with several state-of-the-art invertible sketches.

A. Space and Time Complexities

Our analysis assumes that MV-Sketch is configured with $r = \log\frac{1}{\delta}$, $w = \frac{2}{\epsilon}$, where ϵ ($0 < \epsilon < 1$) is the approximation parameter, δ ($0 < \delta < 1$) is the error probability, and the logarithm base is 2. Theorem 1 states the space and time complexities of MV-Sketch.

Theorem 1. *The space usage is $O(\frac{1}{\epsilon}\log\frac{1}{\delta}\log n)$. The update time per packet is $O(\log\frac{1}{\delta})$, while the detection time of returning all heavy flows is $O(\frac{1}{\epsilon}\log^2\frac{1}{\delta})$.*

Proof. Each bucket of MV-Sketch stores a $\log n$-bit candidate heavy flow and two counters, so the space usage of MV-Sketch is $O(rw\log n) = O(\frac{1}{\epsilon}\log\frac{1}{\delta}\log n)$.

Each per-packet update accesses r buckets and requires $r = \log\frac{1}{\delta}$ hash operations, thereby taking $O(\log\frac{1}{\delta})$ time.

Returning all heavy flows requires to traverse all rw buckets. For each bucket whose $V_{i,j}$ is above the threshold, we check r buckets to obtain the estimate (either $\hat{S}(x)$ or $\hat{D}(x)$) for $x = K_{i,j}$. This takes $O(r^2w) = O(\frac{1}{\epsilon}\log^2\frac{1}{\delta})$ time. □

B. Error Bounds for Heavy Hitter Detection

Suppose that for all flows hashed to a bucket $B(i,j)$, flow x is said to be a *majority flow* of $B(i,j)$ if its sum $S(x)$ is more than half of the total value count $V_{i,j}$. Then Lemma 1 states that the majority flow must be tracked; note that it is a generalization of the main result of MJRTY [7].

Lemma 1. *If there exists a majority flow x in $B(i,j)$, then it must be stored in $K_{i,j}$ at the end of an epoch.*

Proof. We prove by contradiction. By definition, the majority flow x has $S(x) > \frac{1}{2}V_{i,j}$. Suppose that $K_{i,j} \ne x$. Then the increments (resp. decrements) of $C_{i,j}$ due to x must be offset by the decrements (resp. increments) of other flows that are also hashed to $B(i,j)$. This requires that $V_{i,j} - S(x) \ge S(x)$

(i.e., the total value count of other flows is larger than $S(x)$). Thus, $V_{i,j} \geq 2S(x) > V_{i,j}$, which is a contradiction. □

Lemma 2 next bounds the sum $S(x)$ of flow x.

Lemma 2. *Consider a bucket $B(i,j)$ that flow x is hashed to. If $K_{i,j}$ equals x, then $C_{i,j} \leq S(x) \leq \frac{V_{i,j}+C_{i,j}}{2}$; otherwise, $0 \leq S(x) \leq \frac{V_{i,j}-C_{i,j}}{2}$.*

Proof. Suppose that $K_{i,j}$ equals x. Let Δ be the offset amount of x from $C_{i,j}$ due to other flows. Then we have $S(x) = C_{i,j} + \Delta \geq C_{i,j}$. Also, since $V_{i,j} \geq S(x) + \Delta = C_{i,j} + 2\Delta$, we have $\Delta \leq \frac{V_{i,j}-C_{i,j}}{2}$. Thus, $S(x) = C_{i,j} + \Delta \leq \frac{V_{i,j}+C_{i,j}}{2}$.

Suppose now that $K_{i,j} \neq x$. Then the increments (resp. decrements) of $C_{i,j}$ due to x must be offset by the decrements (resp. increments) made by other flows that are also hashed to the same bucket (see the proof of Lemma 1). The total value count of all flows other than x (i.e., $V_{i,j} - S(x)$) minus the offset amount $S(x)$ is at least $C_{i,j}$. Thus, we have $V_{i,j} \geq C_{i,j} + 2S(x)$, implying that $0 \leq S(x) \leq \frac{V_{i,j}-C_{i,j}}{2}$. □

We now study the bounds of the estimated sum $\hat{S}(x)$ of flow x returned by Algorithm 2. From Lemma 2 and the definition of $\hat{S}(x)$ in Algorithm 2, we see that $\hat{S}(x) \geq S(x)$. Also, Lemma 3 states the upper bound of $\hat{S}(x)$ in terms of ϵ and δ.

Lemma 3. *$\hat{S}(x) \leq S(x) + \frac{\epsilon\mathcal{S}}{2}$ with a probability at least $1-\delta$.*

Proof. Consider the expectation of the total sum of all flows except x in each bucket $B(i,j)$. It is given by $E[V_{i,j} - S(x)] = E[\sum_{y \neq x, h_i(y)=h_i(x)} S(y)] \leq \frac{\mathcal{S}-S(x)}{w} \leq \frac{\epsilon\mathcal{S}}{2}$ due to the pairwise independence of h_i and the linearity of expectation. By Markov's inequality, we have

$$\Pr[V_{i,j} - S(x) \geq \epsilon\mathcal{S}] \leq \tfrac{1}{2}. \tag{1}$$

We now consider the row estimate $\hat{S}_i(x)$ (see Algorithm 2). If $K_{i,j}$ equals x, then $\hat{S}_i(x) - S(x) = \frac{V_{i,j}+C_{i,j}}{2} - S(x) \leq \frac{V_{i,j}-S(x)}{2}$ due to Lemma 2; if $K_{i,j} \neq x$, then $\hat{S}_i(x) - S(x) = \frac{V_{i,j}-C_{i,j}}{2} - S(x) \leq \frac{V_{i,j}}{2} - S(x) \leq \frac{V_{i,j}-S(x)}{2}$.

Combining both cases, we have $\Pr[\hat{S}_i(x) - S(x) \geq \frac{\epsilon\mathcal{S}}{2}] \leq \Pr[\frac{V_{i,j}-S(x)}{2} \geq \frac{\epsilon\mathcal{S}}{2}] \leq \frac{1}{2}$ due to Equation (1).

Since $\hat{S}(x)$ is the minimum of all row estimates, we have $\Pr[\hat{S}(x) \leq S(x) + \frac{\epsilon\mathcal{S}}{2}] = 1 - \Pr[\hat{S}(x) - S(x) \geq \frac{\epsilon\mathcal{S}}{2}] = 1 - \Pr[\hat{S}_i(x) - S(x) \geq \frac{\epsilon\mathcal{S}}{2}, \forall i] \geq 1 - (\frac{1}{2})^r = 1 - \delta$. □

Theorem 2 summarizes the error bounds for heavy hitter detection in MV-Sketch.

Theorem 2. *MV-Sketch reports every heavy hitter with a probability at least $1-\delta$ (provided that $\phi\mathcal{S} \geq \epsilon\mathcal{S}$), and falsely reports a non-heavy hitter with sum no more than $(\phi - \frac{\epsilon}{2})\mathcal{S}$ with a probability at most δ.*

Proof. We first prove that MV-Sketch reports each heavy hitter (say x) with a high probability. If flow x is the majority flow in any one of its hashed buckets, it will be reported due to Lemma 1. MV-Sketch fails to report x only if x is not the majority flow of any of its r hashed buckets, i.e., $S(x) \leq \frac{V_{i,j}}{2}$ for $1 \leq i \leq r$ and $j = h_i(x)$. The probability that it occurs (denoted by P) is $P = \Pr[S(x) \leq \frac{V_{i,j}}{2}, \forall i] = \Pr[V_{i,j} - S(x) \geq S(x), \forall i]$. Since $S(x) \geq \phi\mathcal{S} \geq \epsilon\mathcal{S}$, we have $P \leq \Pr[V_{i,j} - S(x) \geq \epsilon\mathcal{S}, \forall i] \leq (\frac{1}{2})^r = \delta$ due to Equation (1). Thus, a heavy hitter is reported with a probability at least $1-\delta$.

We next prove that MV-Sketch reports a non-heavy hitter (say y) with $S(y) \leq (\phi - \frac{\epsilon}{2})\mathcal{S}$ with a small probability. A necessary condition is that y has its estimate $\hat{S}(y) \geq \phi\mathcal{S}$. Thus, $\hat{S}(y) - S(y) \geq \phi\mathcal{S} - (\phi - \frac{\epsilon}{2})S = \frac{\epsilon\mathcal{S}}{2}$. From Lemma 3, we have $\Pr[\hat{S}(y) - S(y) \geq \frac{\epsilon\mathcal{S}}{2}] \leq \delta$. In other words, y is reported as a heavy hitter with a probability at most δ. □

C. Error Bounds for Heavy Changer Detection

Recall that heavy changer detection relies on the upper bound $U(x)$ and the lower bound $L(x)$ of $S(x)$ (see Section III-D). From Lemma 2, both $U(x)$ and $L(x)$ are the true upper and lower bounds of $S(x)$, respectively. Lemma 3 has shown that $U(x)$, which equals $\hat{S}(x)$, differs from $S(x)$ by a small range with a high probability. Now, Lemma 4 shows that $L(x)$ and $S(x)$ also differ by a small range with a high probability.

Lemma 4. *$S(x) - L(x) \leq \epsilon\mathcal{S}$ with a probability at least $1-\delta$.*

Proof. Consider the lower bound estimate $L_i(x)$ given by the hashed bucket $B(i,j)$ of flow x (where $1 \leq i \leq r$) (see Section III-D). If $K_{i,j}$ equals x, $L_i(x) = C_{i,j}$. By Lemma 2, we have $S(x) \leq \frac{V_{i,j}+C_{i,j}}{2}$, implying that $S(x) - L_i(x) = S(x) - C_{i,j} \leq V_{i,j} - S(x)$.

If $K_{i,j} \neq x$, $L_i = 0$ and x is not the majority flow for bucket $B(i,j)$. We have $S(x) - L_i(x) = S(x) \leq V_{i,j} - S(x)$.

Combining both cases, we have $\Pr[S(x) - L(x) \geq \epsilon\mathcal{S}] = \Pr[S(x) - L_i(x) \geq \epsilon\mathcal{S}, \forall i] \leq \Pr[V_{i,j} - S(x) \geq \epsilon\mathcal{S}, \forall i] \leq (\frac{1}{2})^r = \delta$ due to Equation (1). □

Lemma 5 provides an upper bound of the estimated maximum change $\hat{D}(x) = \max\{|U^1(x) - L^2(x)|, |U^2(x) - L^1(x)|\}$ in terms of $\mathcal{S}^1$ and $\mathcal{S}^2$, which are the total sums of all flows in the previous and current epochs, respectively.

Lemma 5. *$\hat{D}(x) \leq D(x) + \epsilon(\mathcal{S}^1 + \mathcal{S}^2)$ with a probability at least $(1-\delta)^2$.*

Proof. Without loss of generality, we consider $\hat{D}(x) = |U^1(x) - L^2(x)|$. Let $S^1(x)$ and $S^2(x)$ be the sums of x in the previous and current epochs, respectively. Let $e_u^1(x) = U^1(x) - S^1(x)$ and $e_l^2 = S^2(x) - L^2(x)$. Then $\hat{D}(x) = |S^1(x) + e_u^1(x) - (S^2(x) - e_l^2(x))| \leq D(x) + e_u^1(x) + e_l^2(x)$. Since $e_u^1(x)$ and $e_l^2(x)$ are independent, we have $\Pr[e_u^1(x) + e_l^2(x) \leq \epsilon(\mathcal{S}^1 + \mathcal{S}^2)] \geq \Pr[e_u^1(x) \leq \epsilon\mathcal{S}^1] \cdot \Pr[e_l^2(x) \leq \epsilon\mathcal{S}^2] \geq (1-\delta)^2$, where the last inequality is due to Lemmas 3 and 4. Thus, $\Pr[\hat{D}(x) - D(x) \leq \epsilon(\mathcal{S}^1 + \mathcal{S}^2)] \geq \Pr[e_u^1(x) + e_l^2(x) \leq \epsilon(\mathcal{S}^1 + \mathcal{S}^2)] \geq (1-\delta)^2$. □

Theorem 3 summarizes the error bounds for heavy changer detection in MV-Sketch.

Theorem 3. *MV-Sketch reports every heavy changer with a probability at least $1-\delta$ (provided that $\frac{\phi\mathcal{D}}{\epsilon} \geq \max\{\mathcal{S}^1, \mathcal{S}^2\}$), and falsely reports any non-heavy changer with change no more than $\phi\mathcal{D} - \epsilon(\mathcal{S}^1 + \mathcal{S}^2)$ with a probability at most $1 - (1-\delta)^2$.*

TABLE I
COMPARISON OF MV-SKETCH WITH STATE-OF-THE-ART INVERTIBLE SKETCHES.

Sketch	r	w	FN Prob.	Space	Update time	Detection time
Count-Min-Heap	$\log \frac{1}{\delta}$	$\frac{2}{\epsilon}$	0	$O(\frac{1}{\epsilon}\log\frac{1}{\delta} + H\log n)$	$\log \frac{H}{\delta}$	$O(H)$
LD-Sketch	$\log \frac{1}{\delta}$	$\frac{2H}{\epsilon}$	0	$O(\frac{H}{\epsilon}\log\frac{1}{\delta}\log n)$	$O(\log \frac{1}{\delta})$	$O(\frac{H}{\epsilon}\log\frac{1}{\delta})$
Deltoid	$\log \frac{1}{\delta}$	$\frac{2}{\epsilon}$	δ	$O(\frac{1}{\epsilon}\log\frac{1}{\delta}\log n)$	$O(\log \frac{1}{\delta}\log n)$	$O(\frac{1}{\epsilon}\log^2\frac{1}{\delta}\log n)$
Fast Sketch	$4H\log\frac{4}{\delta}$	$1+\log\frac{n}{4H\log(4/\delta)}$	δ	$O(H\log\frac{1}{\delta}\log\frac{n}{H\log(1/\delta)})$	$O(\log\frac{1}{\delta}\log\frac{n}{H\log(1/\delta)})$	$O(H\log^3\frac{1}{\delta}\log(\frac{n}{H\log(1/\delta)}))$
MV-Sketch	$\log \frac{1}{\delta}$	$\frac{2}{\epsilon}$	δ	$O(\frac{1}{\epsilon}\log\frac{1}{\delta}\log n)$	$O(\log \frac{1}{\delta})$	$O(\frac{1}{\epsilon}\log^2\frac{1}{\delta})$

Proof. We first prove that MV-Sketch reports each heavy changer (say x) with a high probability. If flow x is the majority flow in any one of its hashed buckets, it must be reported, as its estimate $\hat{D}(x) \geq D(x) \geq \phi\mathcal{D}$. Flow x is not reported only if it is not stored as a candidate heavy flow in both sketches. Since there must exist one sketch (either in the previous or current epoch) with $S(x) \geq \phi\mathcal{D}$, by Theorem 2, the probability that x is not reported in that sketch is at most δ (assuming that $\frac{\phi\mathcal{D}}{\epsilon} \geq \max\{\mathcal{S}^1, \mathcal{S}^2\}$). Thus, a heavy changer is reported with a probability at least $1-\delta$.

We next prove that MV-Sketch reports a non-heavy changer (say y) with $D(y) \leq \phi\mathcal{D} - \epsilon(\mathcal{S}^1 + \mathcal{S}^2)$ with a small probability. Let $\hat{D}(y) = D(y) + \Delta$ for some Δ; hence, $\hat{D}(y) \leq \phi\mathcal{D} - \epsilon(\mathcal{S}^1 + \mathcal{S}^2) + \Delta$. If y is reported as a heavy changer, it requires that $\Delta \geq \epsilon(\mathcal{S}^1 + \mathcal{S}^2)$ and such a probability is at most $1-(1-\delta)^2$ due to Lemma 5. □

D. Error Bounds for Distributed Heavy Flow Detection

We generalize the analysis for a single detector in Theorems 2 and 3 for distributed heavy flow detection under MV-Sketch. Our analysis assumes that the stream of packets of each flow is uniformly distributed to $d \leq q$ detectors.

Theorem 4. *The controller reports every heavy hitter with a probability at least $(1-\delta)^d$, and falsely reports a non-heavy hitter with sum no more than $\frac{d}{q}(\phi - \frac{\epsilon}{2})\mathcal{S}$ with a probability at most $1-(1-\delta)^d$.*

Proof. We first study the probability of reporting each heavy hitter (say x). Recall that the estimate of x at each detector is at least $\frac{\phi}{d}\mathcal{S}$ (see Section IV-B). If all d detectors report flow x to the controller, flow x must be reported as a heavy hitter since its added estimate is at least $d \times \frac{\phi}{d}\mathcal{S} = \phi\mathcal{S}$. Such a probability is at least $(1-\delta)^d$ by Theorem 2.

We next study the probability of reporting a non-heavy hitter (say y). It happens if at least one detector reports flow y to the controller. If $S(y) \leq \frac{d}{q}(\phi - \frac{\epsilon}{2})\mathcal{S}$, the sum of flow y at each detector is at most $\frac{1}{q}(\phi - \frac{\epsilon}{2})\mathcal{S}$. From Theorem 2, the probability that a detector reports flow y is at most δ. Thus, it is falsely reported as a heavy hitter by the controller with a probability at most $1-(1-\delta)^d$. □

Theorem 5. *The controller reports every heavy changer with a probability at least $(1-\delta)^d$, and falsely reports a non-heavy changer with change no more than $\frac{d}{q}(\phi\mathcal{D} - \epsilon(\mathcal{S}^1 + \mathcal{S}^2))$ with a probability at most $1-(1-\delta)^{2d}$.*

Proof. It is similar to that in Theorem 4 and omitted. □

E. Comparison with State-of-the-art Invertible Sketches

We present a comparative analysis on MV-Sketch and state-of-the-art invertible sketches, including Count-Min-Heap [11], LD-Sketch [18], Deltoid [13], and Fast Sketch [24]. Here, we focus on heavy hitter detection in the interest of space. Table I shows the false negative probability, and the space and time complexities, in terms of ϵ, δ, n, and H (the maximum number of heavy hitters in an epoch).

We first study the false negative probability (i.e, the maximum probability of not reporting a heavy hitter); we study other accuracy metrics in Section V. Both Count-Min-Heap and LD-Sketch guarantee zero false negatives as they are configured to keep all heavy hitters in extra structures, while MV-Sketch can miss a heavy hitter with a probability at most δ. Nevertheless, MV-Sketch achieves almost zero false negatives in our evaluation based on real traces (see Section V).

Regarding the space complexity, all sketches have a $\log n$ term. However, it refers to $\log n$ bits (i.e., the key length) in Count-Min-Heap, LD-Sketch, and MV-Sketch, while it refers to $\log n$ integer counters in Deltoid and Fast Sketch.

Regarding the (per-packet) update time complexity, Count-Min-Heap updates the sketch ($O(\log\frac{1}{\delta})$ time) and accesses its heap if the packet is from a heavy flow ($O(\log H)$ time), and its update time increases with H. Both Deltoid and Fast Sketch have high time complexities, which increase with the key length $\log n$. Both MV-Sketch and LD-Sketch have the same update time complexities, yet LD-Sketch may need to expand its associative arrays on-the-fly and this decreases the overall throughput from our evaluation (see Section V).

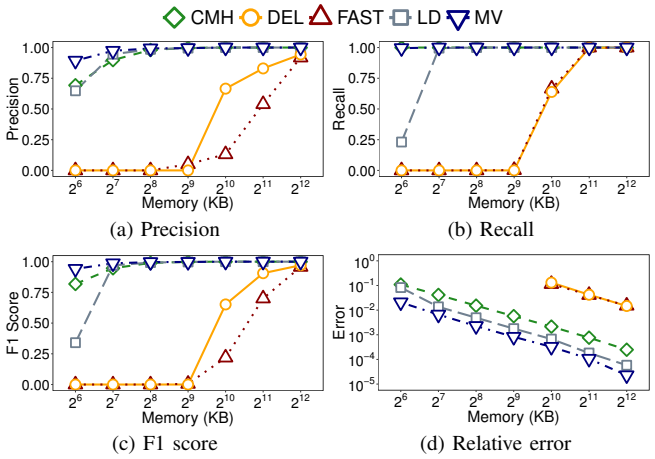
We also present the detection time complexity. However, our evaluation shows that the detection time of recovering all heavy flows is very small (within milliseconds) for all sketches shown in Table I.

V. TRACE-DRIVEN EVALUATION

We show via trace-driven evaluation that MV-Sketch achieves (i) high accuracy in heavy flow detection with small and static memory space, (ii) high processing speed, and (iii) high accuracy in distributed detection, compared to state-of-the-art invertible sketches. We also show how SIMD instructions can further boost the update performance of MV-Sketch.

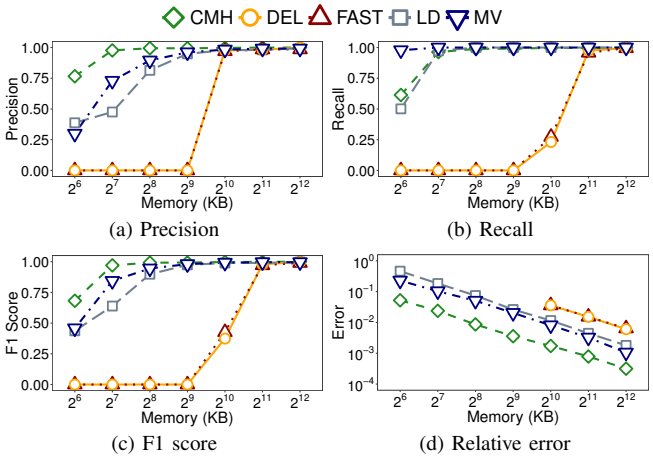
A. Setup

Testbed. We conduct our evaluation on a server equipped with an eight-core Intel Xeon E5-1630 3.70GHz CPU and 16GB RAM. The CPU has 64KB of L1 cache per core, 256KB of L2 cache per core, and 10MB of shared L3 cache. The server runs



(a) Precision (b) Recall (c) F1 score (d) Relative error

Fig. 2. Experiment 1 (Accuracy for heavy hitter detection).



(a) Precision (b) Recall (c) F1 score (d) Relative error

Fig. 3. Experiment 2 (Accuracy for heavy changer detection).

Ubuntu 14.04.5. To exclude the I/O overhead on performance, we load all datasets into memory prior to all experiments.

Dataset. We use CAIDA's anonymized Internet traces [9] captured on an OC-192 backbone link in April 2016. The original traces are one hour long, and we focus on the first five minutes of the traces in our evaluation. We divide the traces into five one-minute epochs and obtain the average results. We measure IPv4 packets only. Each epoch contains 29M packets, 1M flows, and 6M unique IPv4 addresses on average.

Methodology. We take the source/destination address pairs as flow keys (64 bits long). For evaluation purposes, we generate the ground truths by finding $\mathcal{S}$ and $\mathcal{D}$, and hence the true heavy flows, for different epochs. We implement hash functions using MurmurHash [2] in all sketches.

We compare MV-Sketch (MV) with state-of-the-art invertible sketches, including Count-Min-Heap (CMH) [11], LD-Sketch (LD) [18], Deltoid (DEL) [13], and Fast Sketch (FAST) [24].

We consider various memory sizes for each sketch in our evaluation. We fix $r = 4$ and vary w according to the specified memory size. By default, we choose the threshold that keeps the number of heavy flows detected in each epoch as 80 on average. For CMH, we allocate an extra 4KB of memory for its heap data structure to store heavy flows. For LD, since it dynamically expands the associative arrays of its buckets (see Section II-B), we adjust its expansion parameter so that it has comparable memory size to other sketches.

Metrics. We consider the following metrics.

- *Precision*: fraction of true heavy flows reported over all reported flows;
- *Recall*: fraction of true heavy flows reported over all true heavy flows;
- *F1-score*: $\frac{2\times\text{precision}\times\text{recall}}{\text{precision}+\text{recall}}$;
- *Relative error*: $\frac{1}{|R|}\sum_{x,x\in R}\frac{|S(x)-\hat{S}(x)|}{S(x)}$, where R is the set of true heavy flows reported; and
- *Update throughput*: number of packets processed per second (in units of pkts/s).

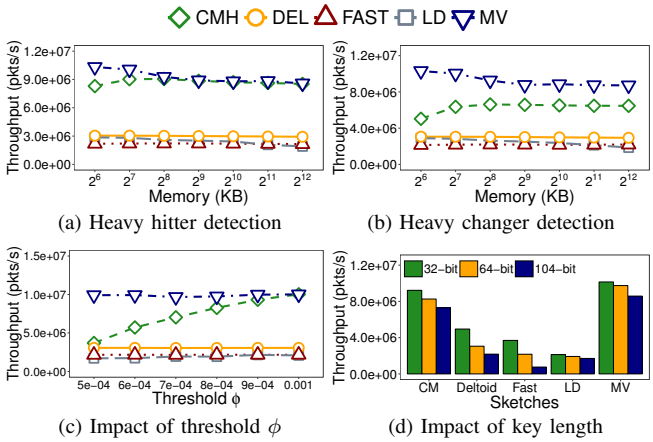
B. Results

Experiment 1 (Accuracy for heavy hitter detection). Figure 2 compares the accuracy of MV-Sketch with that of other sketches in heavy hitter detection. Both DEL and FAST have precision and recall near zero when the amount of memory is 512KB or less, as they need more memory to recover all heavy hitters. Both CMH and LD have high accuracy, except when the memory size is only 64KB, as they do not have sufficient memory to keep all heavy hitters. Overall, MV-Sketch achieves high accuracy; for example, its relative error is on average 55.8% and 87.2% less than those of LD and CMH, respectively.

Experiment 2 (Accuracy for heavy changer detection). Figure 3 compares the accuracy of MV-Sketch with that of other sketches in heavy changer detection. Both DEL and FAST again have almost zero precision and recall when the memory size is 512KB or less. We see that CMH has the highest F1 score and smallest relative error among all sketches, yet its recall is below one for almost all memory sizes. On the other hand, MV-Sketch maintains a recall of one except when the memory size is 64KB, but its precision is low when the memory size is 256KB or less. The reason is that MV-Sketch uses the estimated maximum change of a flow for heavy changer detection, thereby having fewer false negatives but more false positives; we view this as a design trade-off. MV-Sketch achieves both higher precision and recall than LD when the memory size is 128KB or more.

Experiment 3 (Update throughput). We now measure the update throughput of all sketches in different settings. We present averaged results over 10 runs. We omit the error bars in our plots as the variances across runs are negligible.

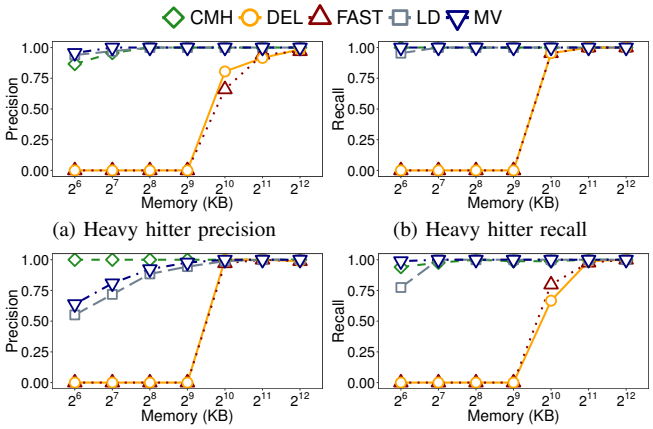
Figure 4(a) shows the update throughput of various sketches in heavy hitter detection. MV-Sketch achieves more than 3× throughput over LD, DEL, and FAST, and 24% higher throughput than CMH when the memory size is 64KB. Note that MV-Sketch (and other sketches as well) sees a throughput drop as the memory size increases, since it cannot be entirely put in cache and the memory access latency increases. The throughput of CMH is much lower than MV-Sketch, especially



(a) Heavy hitter detection (b) Heavy changer detection

(c) Impact of threshold ϕ (d) Impact of key length

Fig. 4. Experiment 3 (Update throughput).



(a) Heavy hitter precision (b) Heavy hitter recall

(c) Heavy changer precision (d) Heavy changer recall

Fig. 5. Experiment 4 (Accuracy for distributed detection).

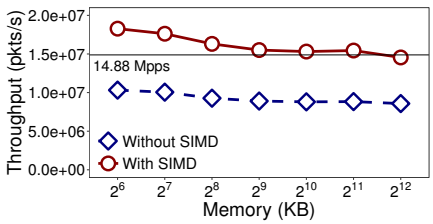


Fig. 6. Experiment 5 (Performance optimizations of MV-Sketch).

when the memory size is 128KB or less, as it sees many false positives and incurs memory access overhead in its heap.

Figure 4(b) shows the update throughput of various sketches in heavy changer detection. MV-Sketch has the highest throughput, which is 1.34–2.05× and 2.98–3.38× over CMH and other sketches, respectively. Note that CMH has lower throughput than in Figure 4(a) although we keep the same number (i.e., 80) of heavy flows in both cases. The reason is that compared to heavy hitter detection, CMH needs to keep more candidates in the heap to guarantee that all heavy changers can be found, thereby incurring higher memory access overhead.

Figure 4(c) shows the impact of the fractional threshold ϕ on the update throughput. Here, we focus on heavy hitter detection and fix the memory size as 64KB. MV-Sketch maintains high and stable throughput (above 9.8M pkts/s) regardless of the threshold value. CMH has slower throughput for smaller ϕ (i.e., more heavy hitters to be detected). For example, when $\phi = 0.0005$, the throughput of CMH is 3.7M pkts/s only. The reason is that the overhead of maintaining the heap increases with the number of heavy flows being tracked.

Figure 4(d) shows the impact of the key length on the update throughput, by setting the flow keys as source addresses (32 bits), source/destination address pairs (64 bits), and 5-tuples (104 bits). We again focus on heavy hitter detection and fix the memory size as 64KB. As the key length increases from 32 bits to 104 bits, the throughput drops of MV-Sketch, CMH, and LD are 15-21%, while those of DEL and FAST are 55-80%. The reason is that the numbers of counters in DEL and FAST increase with the key length, thereby incurring much higher memory access overhead.

Experiment 4 (Accuracy for distributed detection). Figure 5 shows the precision and recall for distributed heavy flow detection, in which we set $d = 3$ and $q = 5$. We observe similar results as in Experiments 1 and 2. Note that we also conduct experiments with different combinations of different settings of d and q, and the results show similar trends.

Experiment 5 (Performance optimizations of MV-Sketch). We make a case that MV-Sketch can leverage SIMD instructions to process multiple data units in parallel and achieve further performance gains. Such performance optimizations enable MV-Sketch to address the need of fast network measurement in software packet processing [17], [23], [25].

Here, we optimize the performance of the Update operation (Algorithm 1). Specifically, we divide a hash value into r parts (where $r = 4$ in our case). We use SIMD instructions to compute the bucket indices of all r rows, load the r candidate heavy flow keys to a register array, and compare the flow key with the r candidate heavy flow keys in parallel. Based on the comparison results, we update the buckets (see Algorithm 1). For 64-bit keys, we use the AVX2 instruction set to manipulate 256 bits (i.e., four 64-bit keys) in parallel.

Figure 6 compares the original and optimized implementations of MV-Sketch. The optimized version achieves 75% higher throughput than the original version on average. Its throughput is above 14.88M pkts/s in most cases, implying that it can match the 10Gb/s line rate (see Section I).

VI. Related Work

Invertible sketches. In Section II-B, we review several invertible sketches for heavy flow detection and their limitations. Another related work extends the Bloom filter [6] with invertibility [14], [16]. In particular, the Invertible Bloom Lookup Table (IBLT) [16] tracks three variables in each bucket: the number of keys, the sum of keys, and the sum of values for all keys hashed to the bucket. To recover all hashed keys, it iteratively recovers from the buckets with only one hashed key and deletes the hashed key of all its associated buckets (so that some buckets now have one hashed key remaining). FlowRadar [23] builds on IBLT for heavy flow detection. However, IBLT

is sensitive to hash collisions: if multiple keys are hashed to the same bucket, it fails to recover the keys in the bucket.

A closely related work to ours is AMON [20], which applies MJRTY in heavy hitter detection. However, AMON and MV-Sketch have different designs: AMON splits a packet stream into multiple sub-streams and tracks the candidate heavy flow for each sub-stream using MJRTY, while MV-Sketch maps each packet to the buckets in different rows in a sketch data structure. MV-Sketch addresses the following issues that are not considered by AMON: (i) providing theoretical guarantees on the trade-offs across memory usage, update/detection performance, and detection accuracy; and (ii) addressing heavy changer detection and distributed detection.

Sketch-based network-wide measurement. Recent studies [17], [19], [23], [25], [27], [31], [32] propose sketch-based network-wide measurement systems for general measurement tasks, including heavy flow detection. Such systems leverage a centralized control plane to analyze measurement results from multiple sketches in the data plane. Our work focuses on a compact invertible sketch design that targets both heavy hitter and heavy changer detection.

Counter-based algorithms. Some approaches (e.g., [4], [5], [15], [21], [26], [30]) track the most frequent flows in counter-based data structures (e.g., heaps and associative arrays), which dynamically admit or evict flows based on estimated flow sizes. They target heavy hitter detection, yet how they work in heavy changer detection and distributed detection remains unexplored.

VII. Conclusion

MV-Sketch is an invertible sketch designed for fast and accurate heavy flow detection. It builds on the majority vote algorithm to enhance memory management in two aspects: (i) small and static memory allocation, and (ii) lightweight memory access in both update and detection operations. It can also be generalized for distributed detection. Trace-driven evaluation demonstrates the throughput and accuracy gains of MV-Sketch. We finally show how the update performance of MV-Sketch can be boosted via SIMD instructions.

Acknowledgments: The work was supported by Research Grants Council of Hong Kong (GRF 14204017), Huawei Technologies (HF2017060008), National Natural Science Foundation of China (61802365), and CAS Pioneer Hundred Talents Program. The corresponding author is Qun Huang.

References

[1] O. Alipourfard, M. Moshref, Y. Zhou, T. Yang, and M. Yu. A Comparison of Performance and Accuracy of Measurement Algorithms in Software. In *Proc. of ACM SOSR*, 2018.

[2] A. Appleby. https://github.com/aappleby/smhasher.

[3] R. B. Basat, G. Einziger, R. Friedman, and Y. Kassner. Heavy Hitters in Streams and Sliding Windows. In *Proc. of IEEE INFOCOM*, 2016.

[4] R. B. Basat, G. Einziger, R. Friedman, and Y. Kassner. Optimal Elephant Flow Detection. In *Proc. of IEEE INFOCOM*, 2017.

[5] R. B. Basat, G. Einziger, R. Friedman, and Y. Kassner. Randomized Admission Policy for Efficient Top-k and Frequency Estimation. In *Proc. of IEEE INFOCOM*, 2017.

[6] B. H. Bloom. Space/time Trade-offs in Hash Coding with Allowable Errors. *ACM Magazine Communications*, pages 422–426, 1970.

[7] R. S. Boyer and J. S. Moore. MJRTY - A Fast Majority Vote Algorithm. In *Automated Reasoning*, pages 105–117. Springer, 1991.

[8] T. Bu, J. Cao, A. Chen, and P. P. C. Lee. Sequential Hashing: A Flexible Approach for Unveiling Significant Patterns in High Speed Networks. *Computer Networks*, 54(18):3309–3326, 2010.

[9] CAIDA. http://www.caida.org.

[10] M. Charikar, K. Chen, and M. Farach-Colton. Finding Frequent Items in Data Streams. In *Proc. of ICALP*, 2002.

[11] G. Cormode and S. Muthukrishnan. An Improved Data Stream Summary: The Count-min Sketch and its Applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[12] G. Cormode and S. Muthukrishnan. Space Efficient Mining of Multigraph Streams. In *Proc. of ACM PODS*, pages 271–282. ACM, 2005.

[13] G. Cormode and S. Muthukrishnan. What's New: Finding Significant Differences in Network Data Streams. *IEEE/ACM Trans. on Networking*, 13(6):1219–1232, 2005.

[14] D. Eppstein and M. T. Goodrich. Straggler Identification in Round-Trip Data Streams via Newton's Identities and Invertible Bloom Filters. *IEEE Trans. on Knowledge and Data Engineering*, 23(2):297–306, 2011.

[15] J. Gong, T. Yang, H. Zhang, H. Li, S. Uhlig, Q. Mary, S. Chen, L. Uden, and X. Li. HeavyKeeper: An Accurate Algorithm for Finding Top-k Elephant Flows. In *Proc. of USENIX ATC*, 2018.

[16] M. T. Goodrich and M. Mitzenmacher. Invertible Bloom Lookup Tables. *arXiv*, 2015. https://arxiv.org/abs/1101.2245.

[17] Q. Huang, X. Jin, P. P. C. Lee, R. Li, L. Tang, Y.-C. Chen, and G. Zhang. SketchVisor: Robust Network Measurement for Software Packet Processing. In *Proc. of ACM SIGCOMM*, 2017.

[18] Q. Huang and P. P. C. Lee. A Hybrid Local and Distributed Sketching Design for Accurate and Scalable Heavy Key Detection in Network Data Streams. *Computer Networks*, 91:298–315, Nov 2015.

[19] Q. Huang, P. P. C. Lee, and Y. Bao. SketchLearn: Relieving User Burdens in Approximate Measurement with Automated Statistical Inference. In *Proc. of ACM SIGCOMM*, 2018.

[20] M. Kallitsis, S. A. Stoev, S. Bhattacharya, and G. Michailidis. AMON: An Open Source Architecture for Online Monitoring, Statistical Analysis, and Forensics of Multi-Gigabit Streams. *IEEE Journal on Selected Areas in Communications*, 34(6):1834–1848, 2016.

[21] R. M. Karp, S. Shenker, and C. H. Papadimitriou. A Simple Algorithm for Finding Frequent Elements in Streams and Bags. *ACM Trans. on Database Systems*, 28(1):51–55, 2003.

[22] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen. Sketch-based Change Detection: Methods, Evaluation, and Applications. In *Proc. of ACM IMC*, 2003.

[23] Y. Li, R. Miao, C. Kim, and M. Yu. FlowRadar: a Better NetFlow for Data Centers. In *Proc. of USENIX NSDI*, 2016.

[24] Y. Liu, W. Chen, and Y. Guan. A Fast Sketch for Aggregate Queries over High-Speed Network Traffic. In *Proc. of IEEE INFOCOM*, 2012.

[25] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman. One Sketch to Rule Them All: Rethinking Network Flow Monitoring with UnivMon. In *Proc. of ACM SIGCOMM*, 2016.

[26] J. Misra and D. Gries. Finding Repeated Elements. *Science of computer programming*, 2(2):143–152, 1982.

[27] M. Moshref, M. Yu, R. Govindan, and A. Vahdat. SCREAM: Sketch Resource Allocation for Software-defined Measurement. In *Proc. of ACM CoNEXT*, 2015.

[28] J. Nelson and D. P. Woodruff. Fast Manhattan Sketches in Data Streams. In *Proc. of ACM PODS*, 2010.

[29] R. Schweller, Z. Li, Y. Chen, Y. Gao, A. Gupta, Y. Zhang, P. Dinda, M. Y. Kao, and G. Memik. Reversible Sketches: Enabling Monitoring and Analysis over High-Speed Data Streams. *IEEE/ACM Trans. on Networking*, 15(5):1059–1072, 2007.

[30] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford. Heavy-Hitter Detection Entirely in the Data Plane. In *Proc. of ACM SOSR*, 2017.

[31] T. Yang, J. Jiang, P. Liu, Q. Huang, J. Gong, Y. Zhou, R. Miao, X. Li, and S. Uhlig. Elastic Sketch: Adaptive and Fast Network-Wide Measurements. In *Proc. of ACM SIGCOMM*, 2018.

[32] M. Yu, L. Jose, and R. Miao. Software Defined Traffic Measurement with OpenSketch. In *Proc. of USENIX NSDI*, 2013.

[33] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker. On the Characteristics and Origins of Internet Flow Rates. In *Proc. of ACM SIGCOMM*, 2002.

[34] Q. Zhao, A. Kumar, and J. Xu. Joint Data Streaming and Sampling Techniques for Detection of Super Sources. In *Proc. of ACM IMC*, 2005.