A High-Performance Invertible Sketch for Network-Wide Superspreader Detection (Supplementary File)

Lu Tang, Yao Xiao, Qun Huang, Patrick P. C. Lee

In this supplementary file, we present (i) the analysis of skewness in different sources of traces, (ii) the proofs of theorems and lemmas, and (iii) additional experimental results.

I. Trace Skewness

In addition to the CAIDA traces collected in the Internet (see the main paper), we also consider two more real-world packet traces from other types of networks. Table I summarizes their details, and Figure 1 shows their skewness. We observe that both UNI2[1] and ENT show high skewness, where the top 3% of sources account for over 90% of total fan-outs. UNI2 and ENT show a similar skewness property as in CAIDA traces. We conduct evaluation results on UNI2 and ENT for different sketches, and observe similar trends (we omit the results here). Thus, we believe that the three CAIDA traces used in the main paper are sufficiently representative.

Table I
Summary of additional traces.

Traces	Description	# unique sources	# distinct pairs
UNI2 [1]	Traces from university data centers	0.032K	1349.16K
ENT [3]	Traces from an enterprise network	0.61K	5485.33K

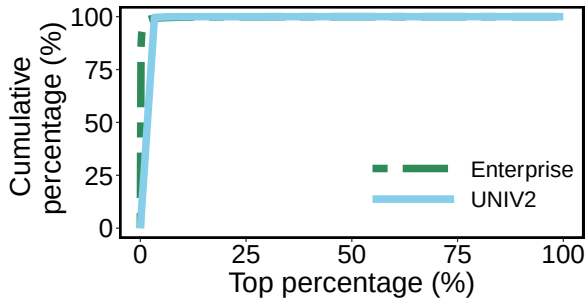


Figure 1. Cumulative fan-out ratios of the top-percentages of sources (i.e., the sum of fan-outs of the top-percentage of sources over the total fan-outs of all sources) for three real-world IP traces.

II. Proofs

We present the proofs of the theorems and lemmas presented in the main paper. Note that the references in the section are based on the main paper.

[1]We do not consider another trace, UNI1, in [1] as the size of the trace is too small to be representative.

A. Proof of Theorem 1

Proof. SpreadSketch has rw buckets, each of which holds an m-bit distinct counter ($V_{i,j}$), a $\log n$-bit candidate source key ($K_{i,j}$), and a $\log\log n$-bit level counter ($L_{i,j}$). Thus, the memory space is $O(rw(m+\log n+\log\log n)) = O(\frac{m+\log n+\log\log n}{\varepsilon}\log\frac{1}{\delta})$.

Each per-packet update takes one hash operation for calculating the level, and then accesses $\log\frac{1}{\delta}$ buckets to update distinct counters. Thus, it takes $O(\log\frac{1}{\delta}+1) = O(\log\frac{1}{\delta})$ time.

SpreadSketch checks all buckets to identify all superspreaders and estimate their fan-outs. It traces at most rw superspreaders, each of which checks r buckets for its fan-out estimation. The detection time is $O(r^2w) = O(\frac{1}{w}\log^2\frac{1}{\delta})$. □

B. Proof of Theorem 2

Proof. For the lower bound, each distinct counter associated with x is updated by at least $S(x)$ times. Given an error factor σ, the estimate returned by each distinct counter is at least $(1-\sigma)S(x)$. As $\hat{S}(x)$ takes the minimum estimate of all distinct counters, we have $\hat{S}(x) \geq (1-\sigma)S(x)$.

For the upper bound, let R_i be the sum of fan-outs of all sources excluding x in the bucket $B(i,h_i(x))$ hashed by x in row i, where $1 \leq i \leq r$. The expectation of R_i, denoted by $E[R_i]$, is given by $E[\sum_{z\neq x, h_i(z)=h_i(x)} S(z)] \leq \frac{\mathcal{S}-S(x)}{w} \leq \frac{\varepsilon\mathcal{S}}{2}$, due to the pairwise independence of h_i and the linearity of expectation. By Markov's inequality,

$$\Pr[R_i \geq \varepsilon\mathcal{S}] \leq \tfrac{1}{2}. \quad (1)$$

Given an error factor σ, we have $\hat{S}(x) \leq (1+\sigma)(R_i+S(x))$ for each row i. Thus, $\Pr[\hat{S}(x) \leq (1+\sigma)(S(x)+\varepsilon\mathcal{S})]$

$= 1-\Pr[\hat{S}(x)-(1+\sigma)S(x) \geq (1+\sigma)\varepsilon\mathcal{S}]$

$\geq 1-\Pr[(1+\sigma)(R_i+S(x))-(1+\sigma)S(x) \geq (1+\sigma)\varepsilon\mathcal{S}, \forall_i]$

$= 1-\Pr[R_i \geq \varepsilon\mathcal{S}, \forall_i] \geq 1-(\frac{1}{2})^r = 1-\delta.$

The theorem follows. □

C. Proof of Lemma 1

Proof. We first describe the idea of our proof. Consider the bucket $B(i,h_i(x))$ hashed by x in row i, where $1 \leq i \leq r$. We partition the distinct source-destination pairs hashed to the bucket into two groups. The first group contains $S(x)$ distinct pairs sharing the same source x, and the second group contains the remaining R_i distinct pairs. Let $M_{S(x)}$ and M_{R_i} be the maximum level values of the two groups, respectively. If $M_{S(x)} > M_{R_i}$, x is stored in $K_{i,h_i(x)}$. Our idea is to show that

if x is a superspreader, then $M_{S(x)} \leq M_{R_i}$ with a very small probability. Let A denote the event $M_{S(x)} \leq M_{R_i}$. Then our problem is to prove that the probability of A, $\Pr[A]$, is at most δ. We prove this in several steps.

Step (i): Deriving the close-form expression of $\Pr[A]$. Let X_t, where $1 \leq t \leq S(x)$, be a random variable that denotes the length of the most significant 0-bits of a hash string for the t-th distinct pair of the source x. X_t follows a geometric distribution with the parameter $\frac{1}{2}$. Thus, we have $\Pr[X_t \leq l] = 1 - \frac{1}{2^{l+1}}$. Recall that $M_{S(x)}$ is the maximum level value among the mutually independent random variables $\{X_t\}_{1 \leq t \leq S(x)}$, we have $\Pr[M_{S(x)} \leq l] = \prod_{1 \leq t \leq S(x)} \Pr[X_t \leq l] = (1 - \frac{1}{2^{l+1}})^{S(x)}$. Similarly, we obtain $\Pr[M_{R_i} \leq l] = (1 - \frac{1}{2^{l+1}})^{R_i}$. Thus,

$$\begin{aligned}
\Pr[A] &= \textstyle\sum_{l \geq 0} \Pr[M_{R_i} = l \text{ and } M_{S(x)} \leq l] \\
&= \textstyle\sum_{l \geq 0} \Pr[M_{R_i} = l] \Pr[M_{S(x)} \leq l] \\
&= \textstyle\sum_{l \geq 0} (\Pr[M_{R_i} \leq l] - \Pr[M_{R_i} \leq l-1]) \Pr[M_{S(x)} \leq l] \\
&= \textstyle\sum_{l \geq 0} ((1 - \frac{1}{2^{l+1}})^{R_i} - (1 - \frac{1}{2^l})^{R_i})(1 - \frac{1}{2^{l+1}})^{S(x)}. \qquad (2)
\end{aligned}$$

Step (ii): Analyzing the upper bound of $\Pr[A]$. Let $R_i = \lambda S(x)$ for some constant $\lambda > 0$. Based on Equation (2), we rewrite $\Pr[A]$ as a function of R_i and λ, denoted by $F(R_i; \lambda)$:

$$F(R_i; \lambda) = \textstyle\sum_{l \geq 0} ((1 - \frac{1}{2^{l+1}})^{R_i} - (1 - \frac{1}{2^l})^{R_i})(1 - \frac{1}{2^{l+1}})^{R_i/\lambda}.$$

We can validate that $F(R_i; \lambda)$ is a decreasing function of R_i and an increasing function of λ.

This helps us simplify $\Pr[A]$ and obtain a rough upper bound. We split R_i into equal-length ranges $I_l = [\frac{lS(x)}{4}, \frac{(l+1)S(x)}{4}]$ for integer $l \geq 0$. Let $P(l) = \Pr[R_i \in I_l]$ for $l \geq 0$. Thus,

$$\begin{aligned}
\Pr[A] &= \textstyle\sum_{l \geq 0} \Pr[A \,|\, R_i \in I_l] P(l) \\
&< \textstyle\sum_{l \geq 0} \Pr[A \,|\, \lambda = \frac{(l+1)}{4}] P(l) \\
&< F(R_i; \tfrac{1}{4}) P(0) + F(R_i; \tfrac{1}{2}) P(1) + \Pr[R_i \geq \tfrac{S(x)}{2}]. \qquad (3)
\end{aligned}$$

To obtain the exact upper bound of $\Pr[A]$, we maximize the right hand side of Inequality (3) by configuring its variables. We first set $P(0)$, $P(1)$, and $\Pr[R_i \geq \frac{S(x)}{2}]$, given the condition that $P(0) + P(1) + \Pr[R_i \geq \frac{S(x)}{2}] = 1$. If x is a superspreader, $S(x) \geq \phi\mathcal{S}$. By the assumption $\varepsilon \leq \frac{\phi}{4}$ and Inequality (1), we have $\Pr[R_i \geq \frac{S(x)}{4}] \leq \Pr[R_i \geq \frac{\phi\mathcal{S}}{4}] \leq \Pr[R_i \geq \varepsilon\mathcal{S}] \leq \frac{1}{2}$; in other words, $P(0) \geq \frac{1}{2}$. Similarly, $\Pr[R_i \geq \frac{S(x)}{2}] \leq \frac{1}{4}$.

Since $F(R_i; \lambda)$ increases with λ, $F(R_i; \frac{1}{4}) < F(R_i; \frac{1}{2}) < 1$. The right hand side of Inequality (3) is maximized when $P(0) = \frac{1}{2}$, $P(1) = \frac{1}{4}$, and $\Pr[R_i \geq \frac{S(x)}{2}] = \frac{1}{4}$. Thus,

$$\Pr[A] < F(R_i; \tfrac{1}{4}) \times \tfrac{1}{2} + F(R_i; \tfrac{1}{2}) \times \tfrac{1}{4} + \tfrac{1}{4}.$$

Step (iii): Quantifying the upper-bound of $\Pr[A]$. Here, we configure some practical values of $S(x)$ to quantify the terms $F(R_i; \frac{1}{4})$ and $F(R_i; \frac{1}{2})$. For example, suppose that $S(x) > 10$. We have $\Pr[A] < 0.269 \times \frac{1}{2} + 0.422 \times \frac{1}{4} + \frac{1}{4} = 0.49 < \frac{1}{2}$.

By considering all r rows, we show that the probability that a superspreader x is not tracked by all r hashed buckets is $\Pr[M_{R_i} \geq M_{S(x)}, \forall_{1 \leq i \leq r}] < \frac{1}{2^r} = \delta$. The theorem follows. □

D. Proof of Theorem 3

Proof. By Theorem 2, $\hat{S}(x) \geq (1-\sigma)S(x) \geq \phi\mathcal{S}$. Then x is not reported as a superspreader if and only if it is not stored in any of its hashed buckets. By Lemma 1, this happens with a probability at most δ. Thus, x is reported as a superspreader with a probability at least $1 - \delta$. □

E. Proof of Theorem 4

Proof. A source x is reported as a superspreader only if $\hat{S}(x) \geq \phi\mathcal{S}$ and x is stored in one of its hashed buckets. We first consider the probability $\Pr[\hat{S}(x) \geq \phi\mathcal{S}]$. From the proof of Theorem 2, we have $\hat{S}(x) \leq (1+\sigma)(R_i + S(x))$ for each row i, where $1 \leq i \leq r$. Thus,

$$\begin{aligned}
\Pr[\hat{S}(x) \geq \phi\mathcal{S}] &\leq \Pr[(1+\sigma)(R_i + \tfrac{\varepsilon\mathcal{S}}{1+\sigma}) \geq \phi\mathcal{S}, \forall_i] \\
&= \Pr[R_i \geq \tfrac{\phi - \varepsilon}{1+\sigma}\mathcal{S}, \forall_i] \\
&\leq \Pr[R_i \geq \varepsilon\mathcal{S}, \forall_i] \text{ (due to } \varepsilon \leq \tfrac{\phi}{4} \text{ and } \sigma < 1) \\
&\leq \tfrac{1}{2^r} = \delta \text{ (by Inequality (1)).}
\end{aligned}$$

We next consider the probability that x is stored in one of its hashed buckets. By Lemma 1, this probability is less than one. Combining both cases, the theorem follows. □

F. Proof of Theorem 5

Proof. From the update operation of HP-SpreadSketch (Figure 5 in the main paper), a pair must be inserted into SpreadSketch at its first appearance (Lines 5-11). Also, the HP-Filter only filters out the repeating pairs, and such repeating pairs do not alter the bucket states in SpreadSketch due to distinct counting in the buckets. Thus, HP-SpreadSketch maintains the same accuracy guarantee as SpreadSketch. □

G. Proof of Theorem 6

Proof. We first quantify the number of packets, X, that belong to the k' non-colliding heavy pairs (denoted by $\mathcal{P}_1$, $\mathcal{P}_2$, $\cdots$, $\mathcal{P}_{k'}$) and are filtered by the HP-Filter (i.e., such packets will not be processed by SpreadSketch). For $1 \leq i \leq k'$, let f_i be the number of packets belonging to the $\mathcal{P}_i$, and e_i be the number of packets that do not belong to $\mathcal{P}_i$ or any of the top-K pairs but are hashed to the same unit of the HP-Filter as $\mathcal{P}_i$. Then $\mathcal{P}_i$ needs at most $e_i + 1$ packets to be considered as the majority pair, and hence at least $f_i - e_i - 1$ packets of $\mathcal{P}_i$ are filtered by the HP-Filter. We have $X \geq \sum_{i=1}^{k'} (f_i - e_i - 1)$.

On expectation, we have $\mathbf{E}[X] \geq \mathbf{E}[\sum_{i=1}^{k'} (f_i - e_i - 1)] = p'N - \sum_{i=1}^{k'} \mathbf{E}(e_i) - k' = p'N - \frac{k'(1-p)N}{t} - k'$.

We further analyze the amortized per-packet update cost of HP-SpreadSketch. Let Y be the total number of packets that are filtered by the HP-Filter (including all heavy and non-heavy pairs). Thus, the amortized per-packet update cost of HP-SpreadSketch is $Y/N + (1 - Y/N)(r+1)$. HP-SpreadSketch has a lower per-packet update cost than SpreadSketch when $Y/N + (1 - Y/N)(r+1) < r$, or equivalently $Y > \frac{N}{r}$.

Note that $Y \geq \mathbf{E}[X]$ (as Y covers all packets from the heavy and non-heavy pairs). If the lower bound of $\mathbf{E}[X]$ is larger

than $\frac{N}{r}$, then we ensure that HP-SpreadSketch has a lower per-packet update cost. The condition means:

$$p'N - \frac{k'(1-p)N}{t} - k' > \frac{N}{r}$$
$$p' > \frac{k'}{N} + \frac{k'(1-p)}{t} + \frac{1}{r}. \quad (4)$$

We can further bound k' in Inequality (4). After the top-K heavy pairs are hashed to the HP-Filter, suppose that the top-K heavy pairs occupy t' out of t units in the HP-Filter. Then there are on expectation $c = \mathbf{E}[k-t'] = k - \mathbf{E}[t'] = k - t[1-(1-1/t)^k]$ hash collisions. In the worst case, the number of colliding pairs is at most twice the number of hash collisions (i.e., $2c$). Then we have $k' \geq k - 2c = 2t[1-(1-1/t)^k] - k$. Combining it with Inequality (4), we obtain the new inequality in which HP-SpreadSketch has a lower per-packet update cost.

$$p' > \frac{2t[1-(1-1/t)^k]-k}{N} + \frac{\{2t[1-(1-1/t)^k]-k\}(1-p)}{t} + \frac{1}{r}. \quad (5)$$

□

H. Proof of Theorem 7

Proof. Compared with SpreadSketch, HP-SpreadSketch has an additional HP-Filter with t units, each of which holds a $\log 2n$-bit candidate majority pair (recall that the key size is $\log n$ bits) and a 32-bit indicator counter. Thus, the total memory space is $O(rw(m + \log n + \log\log n) + t(\log 2n + 32)) = O(\frac{m+\log n+\log\log n}{\varepsilon} \log\frac{1}{\delta} + t\log n + t)$.

In the worst case, each per-packet update has one hash operation for the HP-Filter and has the same insertion cost in SpreadSketch, so it takes $O(1 + \log\frac{1}{\delta} + 1) = O(\log\frac{1}{\delta})$ time.

HP-SpreadSketch has the same detection procedure as SpreadSketch, so its detection time is also the same as in SpreadSketch. □

III. Additional Experiments

(Experiment S1) SpreadSketch with different distinct counters. We evaluate the effectiveness of using the multiresolution bitmap [7] in SpreadSketch. We consider four different distinct counters, including Linear Counting (LC) [9], K-Minimum Values (KMV) (with $K = 4$) [5], HyperLogLog (HLL) [8], and the multiresolution bitmap (SS) [7]. We compare their accuracy as well as performance.

We first fix the r and w of SpreadSketch and vary the memory size of each type of distinct counter. Figure 2 shows the results. We observe that the multiresolution bitmap is the only distinct counter that achieves the stable accuracy for all three traces in almost all the settings. Its F1-score is above 0.75 in all cases except for the memory size of 32 bits for CAIDA16 and CAIDA18. It also has the smallest relative error in most of the cases among all types of distinct counters.

We further fix the memory size of each distinct counter as 438 bits (as used in our implementation of SpreadSketch) and vary w of SpreadSketch. Figure 3 shows the results. We make the similar conclusion that the multiresolution bitmap maintains the stable and highest accuracy among all types of distinct counters.

(Experiment S2) KMV in superspreader detection. We examine K-Minimum Values (KMV) [5] in detail and evaluate the accuracy and performance of using KMV as the distinct counter in SpreadSketch. We focus on the impact of K in KMV, where K stands for the number of the smallest hash values used for distinct counter estimation. Specifically, when $K > 1$, we calculate the fan-out from KMV with the unbiased estimator $(K-1) \times \text{HASH\_MAX}/H_{(K)}$, where $H_{(K)}$ is the k-th smallest hash value we have seen and HASH_MAX is the maximum value that the hash function can return; when $K = 1$, we estimate the fan-out with the basic estimator $K \times \text{HASH\_MAX}/H_{(K)}$. A larger K implies more accurate estimation, at the expense of more memory and computational overhead.

We first fix the number of rows (i.e. r) and the number of buckets in each row (i.e., w) of the sketch, and vary K in KMV. We use the same threshold as in Experiment 1. Figure 4 shows the results. We observe that the F1-score of KMV shows an increasing trend as K increases. However, it is below 0.55 for the less skewed trace CAIDA16 in all settings (Figure 4(c)). We also observe a dip in $K = 2$ for the F1-score. The reason is that when the estimator changes from $K \times \text{HASH\_MAX}/H_{(K)}$ for $K = 1$ to $(K-1) \times \text{HASH\_MAX}/H_{(K)}$ for $K = 2$, the estimation of fan-out drops and hence causes an increasing number of false negatives (Figure 4(b)). The throughput of KMV drops as K increases, and is below 14.88 MPPS for $K \geq 7$ (Figure 4(e)).

We then fix the total memory usage at 2 MiB and vary K. Specifically, we fix $r = 4$. Given K, we infer the memory size of each bucket, and vary w such that the total memory usage is at 2 MiB. Thus, a larger K implies fewer buckets in each row for the sketch. Figure 5 shows the results. We observe that the F1-score of KMV on CAIDA16 and CAIDA18 increases first and decreases later as K increases (Figure 5(c)). The reason is that when K is large, the error is mainly attributed to the sketch with fewer buckets. The throughput of KMV shows a decreasing trend as K increases (Figure 5(e)).

(Experiment S3) Comparisons between SpreadSketch with BeauCoup. We compare SpreadSketch with BeauCoup [6] in accuracy and performance versus the memory usage. BeauCoup is a sampling-based approach designed to support multiple distinct counting queries simultaneously with a small constant number of memory accesses per packet. It counts the fan-out of each source with a 32-bit bitmap. We reproduce the implementation of BeauCoup in C++. Our current implementation supports only a single query for superspreader detection. As the configuration of BeauCoup depends on the exact threshold value, we need to calculate the exact threshold value based on the fraction threshold ϕ and the total fan-outs in each epoch. We then configure BeauCoup subject to the exact threshold value via the procedure for the configuration described in its original paper (Section 3.2 in [6]). Figure 6 shows the results. Compared with BeauCoup, SpreadSketch achieves higher F1-score and reduces the relative error by more than 66%. The throughput of BeauCoup is much higher than that of SpreadSketch, as it is based on sampling while SpreadSketch needs to update multiple rows of the sketch structure for every packet.

(Experiment S4) Throughput of SpreadSketch with different memory settings. We evaluate the update throughput of SpreadSketch versus the memory usage on an Intel i7-10700

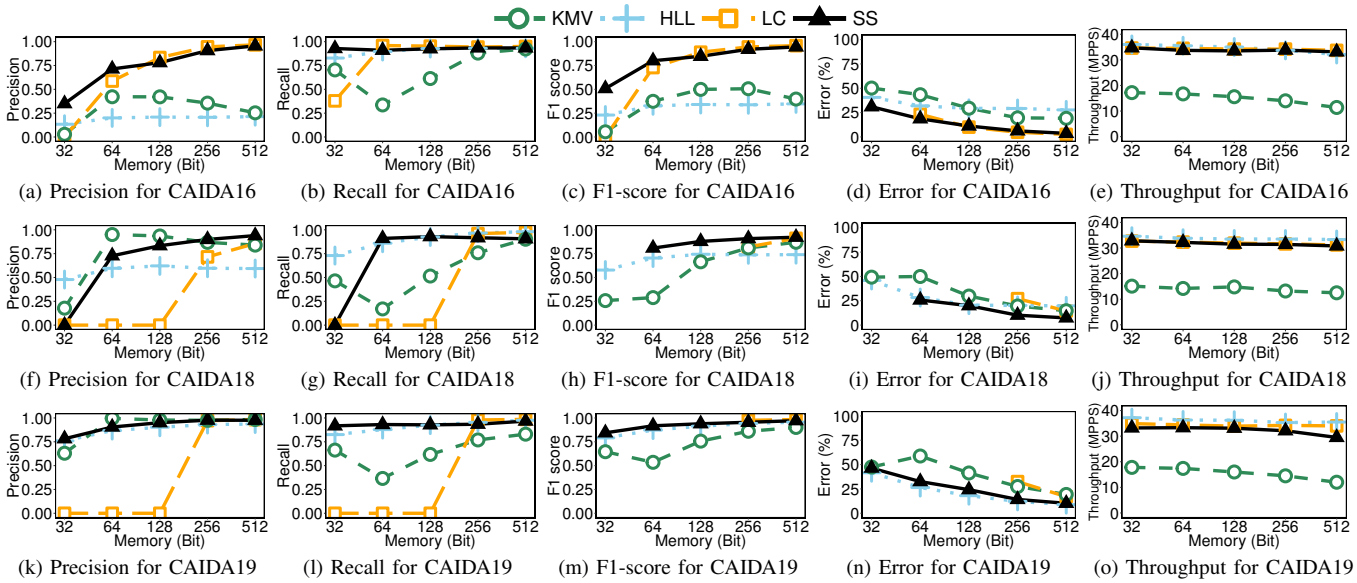


Figure 2. (Experiment S1) Impact of distinct counters on SpreadSketch versus the memory size of a distinct counter.

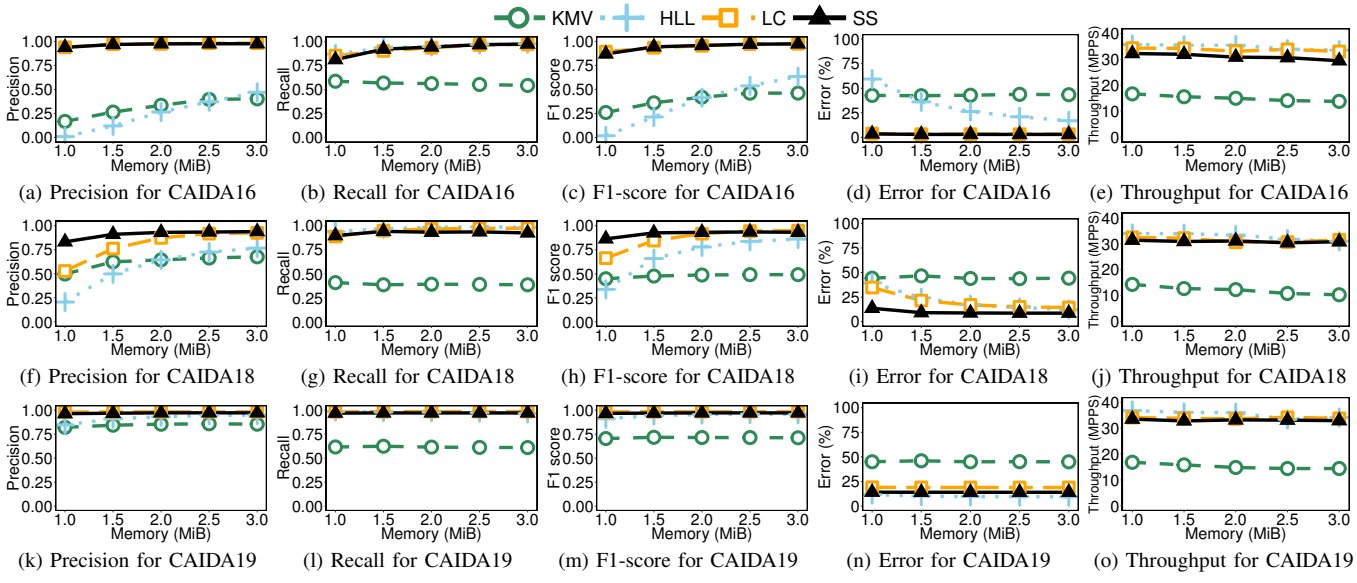


Figure 3. (Experiment S1) Impact of distinct counters on SpreadSketch versus the memory size of SpreadSketch.

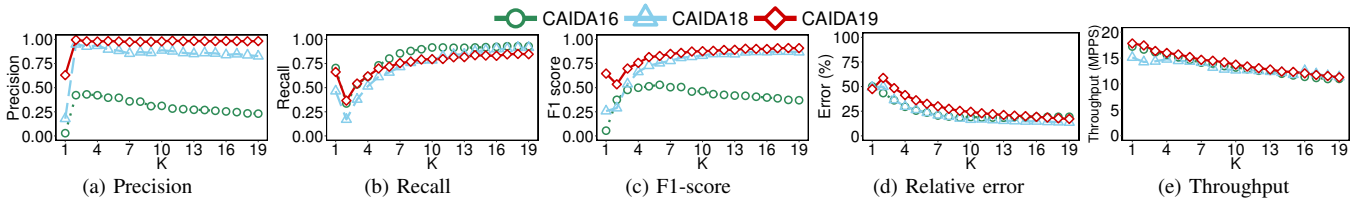


Figure 4. (Experiment S2) KMV in superspreader detection, where we fix r and w of sketch and vary K.

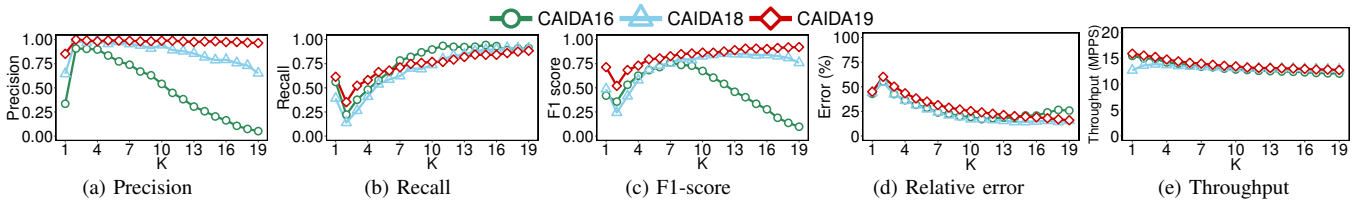
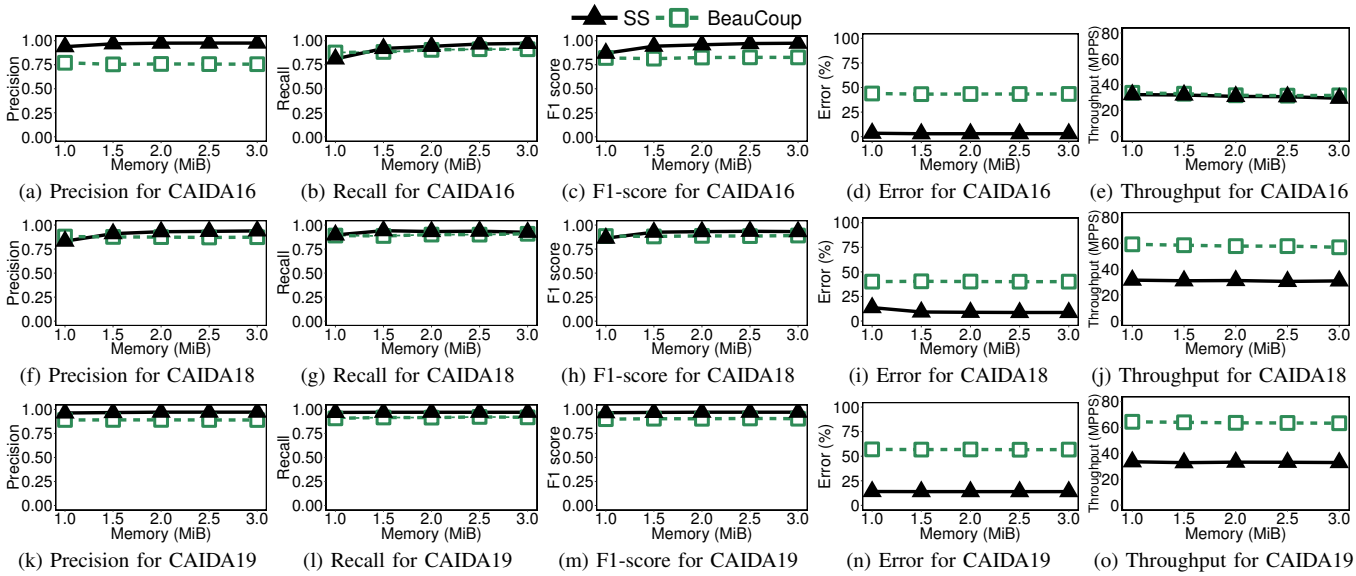


Figure 5. (Experiment S2) KMV in superspreader detection, where we fix the total memory and vary K.

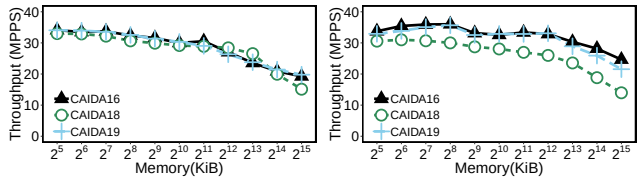


(a) Precision for CAIDA16 (b) Recall for CAIDA16 (c) F1-score for CAIDA16 (d) Error for CAIDA16 (e) Throughput for CAIDA16

(f) Precision for CAIDA18 (g) Recall for CAIDA18 (h) F1-score for CAIDA18 (i) Error for CAIDA18 (j) Throughput for CAIDA18

(k) Precision for CAIDA19 (l) Recall for CAIDA19 (m) F1-score for CAIDA19 (n) Error for CAIDA19 (o) Throughput for CAIDA19

Figure 6. (Experiment S3) Comparisons between SpreadSketch and BeauCoup.



(a) Throughput for SpreadSketch (b) Throughput for HP-SpreadSketch

Figure 7. (Experiment S4) Throughput of SpreadSketch with different memory settings.

processor with 8 cores, 64 KiB of L1 cache per core, 256 KiB of L2 cache per core, and 16 MiB of shared L3 cache. We vary the memory usage of SpreadSketch from 32 KiB to 32 MiB, so that the memory allocation can cover all cache levels. Figure 7 shows the results. We observe that the update throughput of both SpreadSketch and HP-SpreadSketch decreases as the memory usage increases. The reason is that the working sets of SpreadSketch and HP-SpreadSketch cannot fit in cache with large memory usage. Thus, both schemes need more CPU cycles to fetch data into cache, thereby increasing the delay for packet processing. Although the performance of SpreadSketch degrades when the cache size is insufficient, we can mitigate the degradation by performing extensive system-level optimization [4], such as using DPDK [2] to read packets with kernel-bypass and adopting Single Instruction Multiple Data (SIMD) instructions to enable instruction-level parallelism across packets.

REFERENCES

[1] Data set for IMC 2010 data center measurement. https://pages.cs.wisc.edu/~tbenson/IMC10_Data.html.

[2] DPDK. https://www.dpdk.org/.

[3] LBNL/ICSI enterprise tracing project. http://www.icir.org/enterprise-tracing/.

[4] 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*, pages 1–14, 2018.

[5] K. Beyer, P. J. Haas, B. Reinwald, Y. Sismanis, and R. Gemulla. On Synopses for Distinct-Value Estimation Under Multiset Operations. In *Proc. of the ACM SIGMOD*, pages 199–210, 2007.

[6] X. Chen, S. Landau-Feibish, M. Braverman, and J. Rexford. BeauCoup: Answering many network traffic queries, one memory update at a time. In *Proc. of ACM SIGCOMM*, pages 226–239, 2020.

[7] C. Estan, G. Varghese, and M. Fisk. Bitmap algorithms for counting active flows on high speed links. In *Proc. of ACM IMC*, 2003.

[8] P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier. Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm. In *Analysis of Algorithms*, 2007.

[9] K.-Y. Whang, B. T. Vander-Zanden, and H. M. Taylor. A linear-time probabilistic counting algorithm for database applications. *ACM Trans. on Database Systems*, 15(2):208–229, 1990.