Diamond Sketch: Accurate Per-flow Measurement for Big Streaming Data

Tong Yang, Siang Gao, Zhouyi Sun, Yufei Wang, Yulong Shen, and Xiaoming Li

Abstract—Per-flow measurement is a critical issue in computer networks, and one of its key tasks is to count the number of packets in each flow (for big streaming data). The literature has demonstrated that *sketch* is the most memory-efficient data structure for the counting task, and is widely used in distributed systems. Existing sketches often use many counters that are of the same size to record the number of packets in a flow, thus the counters are forced to be large enough to accommodate the size of the largest flow. Unfortunately, as most flows are small (*i.e.*, mice flows) and only a very few flows are large (*i.e.*, elephant flows), many counters represent very small values, which is a waste of memory. Sketches are often stored in fast but expensive memory (*e.g.*, SRAM), thus it is critical to achieve high memory efficiency. To address this issue, we propose a novel sketch, namely the Diamond sketch. The Diamond sketch is composed of atom sketches, and each atom sketch uses small counters. The key idea of Diamond is to *dynamically assign an appropriate number of atom sketches to each flow on demand, thus optimizing memory efficiency*. Experimental results show that the Diamond sketch outperforms the best of the five typical sketches by up to 508.3 times in terms of relative error while keeping comparable speed. We made the source code of all the six sketches available on GitHub [1].

Index Terms—sketch, data streams, accuracy, distributed monitoring.

1 Introduction

1.1 Background and Motivation

PER-FLOW measurement is a critical issue in computer networks, and it provides information for anomaly detection, capacity planning, accounting and billing, and service provising [2], [3], [4], [5], [6]. In per-flow measurement, one fundamental problem is to estimate the number of packets in each flow (flow size) for big streaming data. A flow is often identified by a certain combination of fields in the five-tuple in the packet's header: source IP address, destination IP address, source port number, destination port number, and protocol type. Network data in one second contains hundreds of millions of flows. Real network flows have **two characteristics**: *high-speed* and *non-uniform distribution*. On the one hand, the speed of network traffic is so high that it is very hard to make a precise record of sizes of flows [7]. On the other hand, the sizes of network flows are usually non-uniformly distributed [8], [9], [10], which means a small part of flows have very large sizes (*elephant flows*) while most flows have small sizes (*mice flows*). Two typical distributions of this kind of network traffic are Zipfian [11] and Powerlaw [12], and the flows whose sizes follow Zipfian distribution are called *skewed* network traffic.

Due to the first characteristic of network traffic – high speed, approximately recording and estimating data with sketches has gained popularity [9], [10], [13]. Sketches are probabilistic data structures, and can achieve small memory footprints, high accuracy, and fast speed of insertions and queries. Sketch plays an important role in distributed system measurement, because it can perform network wide measurement for the whole system [14], [15], [16]. In a classical scenario, a data center has many monitoring nodes, each running a sketch to monitor the network flows. And a collector will collect all the data from those sketches and produce useful information. Existing sketches work well on uniform traffic. Unfortunately, they cannot work well in practice because of the second characteristic of network traffic – non-uniform distribution. Specifically, conventional sketches (such as CM [13], CU [17], Count [18], CSM [19] sketches) use counters that are of the same size to store the number of packets. On the one hand, elephant flows are often considered to be more important than mice flows, thus the counters need to be large enough to represent the largest size of the elephant flows. On the other hand, the large quantity of mice flows means that most counters will only represent a small value, and the higher bits in these counters are all 0, leading to a waste of memory. Furthermore, the memory usage is limited for sketches, because in order to catch up with the high speed of network traffic, the sketch should be stored in fast memory (such as SRAM, Static RAM) [7]. Compared to DRAM (Dynamic RAM), SRAM is tens of times faster, but is much more expensive and limited in size [20]. Therefore, limited memory size and big counters lead to a limited number of counters, which further incurs a high collision rate in sketches, and finally drastically degrades the accuracy of sketches. The `goal` of this paper is to design a novel sketch that can *achieve a much higher accuracy than the prior art, especially when processing non-uniformly distributed network traffic using just a small memory footprint*.

- *T. Yang, S. Gao, Z. Sun, Y. Wang and X. Li are with the School of EECS, Peking University, No.5 Yiheyuan Road, Haidian District, Beijing, P.R. China. E-mail: yangtongemail@gmail.com, gao.siang@pku.edu.cn, sunzhouyi@pku.edu.cn, wang.yufei@pku.edu.cn, lxm@pku.edu.cn.*
- *Y. Shen is with the School of Computer Science and Technology, Xidian University, Xian 710071, China. E-mail: ylshen@mail.xidian.edu.cn.*

1.2 Proposed Approach

In this paper, we propose a new sketch, namely the Diamond sketch, as it takes on a diamond shape. A Diamond sketch is composed of many small sketches, whose counters have a very small number of bits (*e.g.*, 4). Each small sketch is called a *carbon atom sketch* or an *atom sketch*, because a Diamond sketch consists of many carbon atom sketches, similar to that a diamond is composed of many carbon atoms.

The **key idea** of Diamond is to assign atom sketches to flows *on demand*. A Diamond consists of an increment part, a carry part and a deletion part. The increment part, along with the carry part, is used for insertion and query, and the deletion part is used for deletion. These three parts are all composed of atom sketches: the increment part contains several atom sketches, while the carry part and the deletion part both contain only one atom sketch.

We recommend using the CU sketch [17] as the atom sketch, because of its easy implementation, high efficiency, and high accuracy. Here we briefly show how the CU sketch works. Specifically, the CU sketch is an array A with w counters and k independent hash functions $h_i(.)$ $(1 \leqslant i \leqslant k, 1 \leqslant h_i(.) \leqslant w)$. Given an incoming packet, first, it obtains the flow ID from the packet's header, computes k hash functions and locates k mapped counters. Then it increments only the smallest counter(s) by 1. There could be more than one smallest counters, and we need to increment all the smallest counters by 1. When querying a flow ID, it computes k hash functions, and returns the minimum value of the k mapped counters.

As a matter of fact, the atom sketch could be any common sketch mentioned in this paper. For example, if we use the CM sketch as our atom sketch, due to the similarity between the CM sketch and the CU sketch, the algorithms of insertion and query remain the same. As for the deletion part, we do not even need it as the CM sketch supports deletions itself, and thus the memory usage can be further reduced. However, because deletion operations are far less frequent than insertions in network measurement, and using CU sketches as atom sketch achieves higher accuracy, we will mainly discuss the data structure using the CU sketch as the atom sketch.

The Diamond sketch is composed of three parts: *the increment part, the carry part,* and *the deletion part*. Each part is composed of one or several atom sketches.

Increment part: As shown in Figure 1, the increment part of our Diamond sketch is made up of d atom sketches: $I_1, I_2, ..., I_d$, where I_j has L_j counters. j is called the *depth* of sketch I_j, and d is called the *depth* of our Diamond sketch. Each counter of all these atom sketches has w_1 bits. w_1 is usually very small (*e.g.* $w_1 = 4$), thus overflows may happen frequently. To record an incoming packet, we first insert it into the first atom sketch I_1. If the corresponding counters in I_1 overflow, we will insert it into I_2, and so on and so forth, until there is no overflow or counters in I_d overflow.

Carry part: To record the overflow status of each flow, we also need a carry part (an atom sketch $\mathcal{C}$, as shown in Figure 1) to tell the deepest sketch a flow has been inserted into. Suppose a flow causes overflows in atom sketch I_1 and I_2 when being inserted, and is successfully inserted into I_3, 3 is called the *overflow depth* of the flow. We query this flow in $\mathcal{C}$. If the query result is 2, which means this flow is inserted into I_3 for the first time, we need to update the carry part by inserting this flow into $\mathcal{C}$. Otherwise, we do not update the carry part. The carry part is also needed during query operations. When querying a flow, we first query it in $\mathcal{C}$. Suppose the result is a, which means we need to query this flow in $I_1, I_2, ..., I_a$ to calculate the query result.

Deletion part: Per-flow measurement usually does not require deletions, since the size of each flow will never decrease as the network traffic passes by. Despite that deletions are not always necessary, in some scenarios requiring deletions (*e.g.*, when a flow ends, some applications need to delete its information from the sketch), Diamond can support this operation by adding a deletion part, which is also an atom sketch.

Apart from per-flow measurement, Diamond can also be applied in other important issues in computer networks, such as top-k problems and multiple-set membership query problems.

1.3 Key Contributions

- We propose a novel sketch, namely Diamond, to accurately and quickly record and query the sizes of flows in real network traffic.
- We use the Diamond sketch to address another two important network issues: top-k and multiple-set membership query.
- We have conducted extensive experiments, and results show that our Diamond sketch outperforms the best of the prior art by two orders of magnitude in terms of relative error in flow size estimtation, and also significantly outperforms the state-of-the-art solutions when handling the problems of top-k and multiple-set membership query.

2 Related Work

To address the per-flow measurement problem, three kinds of data structures have been proposed: sketches, Bloom filters, and counters. More details are provided in the survey [9].

Sketches: We choose five most popular sketches in this paper, namely CU [17], Count [18], CSM [19], Count-min [13], and Augmented [10] sketches, and compare them with Diamond. We have already introduced the CU sketch in Section 1.2, and the other four sketches will be introduced as follows.

The Count-min (CM) sketch is almost the same as the CU sketch, except that during insertions, it adds one to all the k mapped counters, instead of the smallest one(s) among them. In this way, the CM sketch can support deletion.

As for conventional methods like the CU sketch, a counter with small size will unexpectedly overflows which could lead to the loss of accuracy especially for large flows. However, in most scenarios, the measurement of large flows is the most crucial part, such as finding heavy hitters, finding heavy changes, finding SuperSpreaders, and finding network anomaly. To cut the size of the counters of the CU sketch would limit its usage.

In contrast, since our data structure allocates counters more reasonably for large flows and small flows, it can achieve a high accuracy under different scenarios while consuming a small memory usage.

The structure of the Count (C) sketch [18] is exactly the same as the CM sketch, except that in addition to the k hash functions $h_i(.)$ $(1 \leqslant i \leqslant k)$, the sketch is also associated with another k hash functions $g_j(.)$ $(1 \leqslant j \leqslant k)$, each of which hashes a flow ID to -1 or 1 with equal probability. To record a packet with flow ID e, the Count sketch first calculates $h_i(e)$ and $g_i(e)$ $(1 \leqslant i \leqslant k)$ and adds $g_i(e)$ to counter $A[h_i(e)]$. When querying an flow with ID e, it reports the median of $\{A_i[h_i(e)] \cdot g_i(e),\ (1 \leqslant i \leqslant k)\}$ as the estimated query result.

The CSM sketch [19] computes k hash functions when a packet with flow ID e is inserted, but instead of incrementing all k mapped counters, CSM sketches randomly increment one of them. And during the query process, CSM sketches will add up the values in the k mapped counters, and return this result subtracted by a noise value.

The Augmented (A) sketch [10] is a sketch framework built upon other sketches, like CM sketches or CU sketches. It adds an additional filter to the existing sketches. To record a packet with flow ID e, if e is already in the filter, A sketch just updates its size in the filter; otherwise, it inserts it on the sketch using the standard insertion operation of that sketch, and queries e in the sketch. If the query result of e is larger than the size of e' which has the smallest size in the filter, it replaces e' with e and expels e' into the sketch. This technique will make the query results of only those flows residing in the filter (which are usually elephant flows) more accurate, at the cost of a loss in speed of queries and insertions.

Bloom filters variants: The original Bloom filter [21] can tell whether a flow has appeared or not. Later several variants of the Bloom filter made enhancements on the original one to let it be able to store the size of a flow instead of merely determining its appearance. Those variants include Counting Bloom filters (CBF) [22], Spectral Bloom filters (SBF) [23], and Dynamic Count Bloom filters (DCF) [24].

There are several Bloom filter variants for frequency estimation, such as SBF and DCF, which use complicated techniques with many pointers to improve accuracy at the cost of slow update. What is worse, they cannot be implemented in hardware. We aim to achieve both higher accuracy, comparable speed, and easy-to-implement in both software and hardware. Therefore, We do not conduct experiments to compare the performance of Diamond with these variants.

Counter variants: The Counter Braids [25] algorithm performs compression while counting, and can recover the compression result almost errorlessly. However, the author admits that it has to know the IDs of all flows beforehand and does not support instantaneous point query. Furthermore, the CSM sketch performs better than the Counter Braids in terms of accuracy according to literature [19]. Therefore, the Counter Braids algorithm is not included in our experiments.

The key idea exploited in this paper may seem to be similar to that of [26] (dividing counters into two groups), but they are actually very different. Our paper mainly focuses on reducing the memory usage of the data structure to accommodate it to small and expensive SRAM. There is no dichotomy of storage, but different layers of sketches whose sizes decrease gradually. Also, the process of allocating more counters for large flows is completed by natural overflows. In contrast, [26] focuses on balancing the number of counters and the rate of updating between DRAM and SRAM. Recent updates are held by the SRAM counters and is transferred to the DRAM counterparts periodically. The algorithm in [26] is a rough sorting algorithm, organizing counters into high and low-order bins according to a threshold derived from formal analyses.

When we only use the CU sketch, to avoid overflow of counters for large flows which are often very important, every counter should be large enough. In this way, for counters that are mapped by small flows, their higher bits will all be 0 and wasted. To minimize such a memory waste, Diamond leverages natural overflows and reduces size of deeper layers of the increment part, so as to automatically allocate more atom sketches for large flows and fewer ones for small ones.

3 The Diamond Sketch

3.1 Rationale

As mentioned in Section 1.1, real datasets are often non-uniform or skewed. Existing sketches will achieve lower accuracy for such datasets. To address this issue, we propose the conception of **carbon atom sketch** (atom sketch for short), which is a sketch composed of small counters (*e.g.*, 4 bits). We will start recording items with one atom sketch I_1. Since its counters are small, overflows may happen frequently. When overflows happen, we use the second atom sketch I_2 to help record the value that cannot be represented by one single small counter. I_2 may also overflow, and we can subsequently find another one I_3, and so on. In this way, an item with a large value will be recorded by several small counters from several atom sketches. Obviously, the number of items inserted into I_{i+1} is smaller than that inserted into I_i. Therefore, to achieve memory efficiency, the size of I_{i+1} should be smaller than I_i. Furthermore, those counters located in deeper layers of the increment part will only be accessed by large flows through the process of overflowing during insertions. In contrast, small flows would mostly stay in the surface of the increment part, *e.g.*, the first layer. In other words, we complete the task of assigning an appropriate number of atom sketches for each flow according to its size naturally in the process of insertion. What is more, when it comes to uneven distributions, the more skewed the flow size distribution is, the longer and smaller their "tails" will be, and the more likely the tails (small flows) will be stored in the first layer, and large flows will be stored across multiple layers. In this way, we can achieve high accuracy for both large and small flows.

To compute the desired value when querying an item, we also need to record the `overflow depth` of each item during insertions. Specifically, when inserting an item e, if it is inserted into I_1, I_2, ... I_i, we need to record its `overflow depth` i. There are many ways to do that, such as adding an extra bit to each counter to indicate whether it has overflowed, or using a Bloom filter for each atom sketch to record the items suffering overflows. We finally choose to use another atom sketch $\mathcal{C}$ to record the overflow depth of each item, because of the following two reasons. First, the sketch can be more accurate than Bloom filters and flag bits when using the same size of memory. Second, we use another atom sketch to record the overflow depth, rather than introduce heterogeneous data structures, so as to keep the homogeneity of our Diamond.

Our sketch can be extended to support deletions when needed in some application scenarios. As mentioned before, we recommend using the CU sketch as the atom sketch, but the CU sketch does not support deletions. To address this issue, we propose to use another atom sketch $\mathcal{D}$ to record the frequency that each item has been deleted. When querying an item e, we subtract the value reported by the deletion part from that reported by the increment and carry part as the estimated frequency of e.

In sum, our sketch is composed of three parts: increment part, carry part, and deletion part. Increment part is composed of several atom sketches, while carry part and deletion part are both composed of one atom sketch. We name our sketch **Diamond sketch**, because our whole sketch is composed of atom sketches, which is like a diamond is composed of carbon atoms.

By layering up the atom sketches of the increment part, most counters in the deep layers will be preserved primarily for storing large flows, and small flows will be mostly recorded in the shallow layers. This is equivalent to a dynamic allocation of atom sketches when inserting flows of different sizes. This key feature of Diamond sketch enables a efficient use of memory, and further guarantees high accuracy as shown in later experiments. Moreover, the design to gradually decrease the size of layers in the increment part avoids the potential waste of memory in deep layers. At last, the error rate of the carry part is practically negligible, as values stored in it are indeed very small (no more than the depth of the increment part), which further ensures the overall accuracy of the Diamond sketch.

3.2 Data Structure

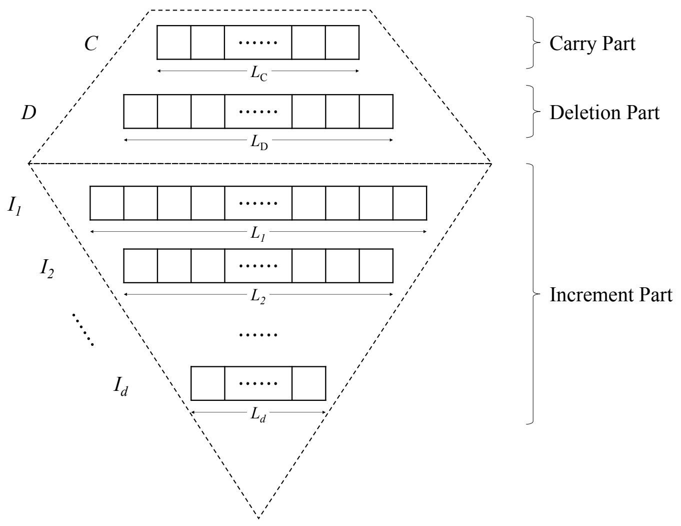


Fig. 1. Data structure of the Diamond sketch.

As shown in Figure 1, **Increment part** together with carry part records the size of each flow that has been inserted. It consists of d atom sketches, where we denote the i^{th} atom sketch with I_i. Each atom sketch I_i is composed of L_i counters, and each counter contains w_1 bits. We denote the j^{th} counter of the i^{th} atom sketch with $I_i[j]$. $\{L_1, L_2, ..., L_d\}$ is a decreasing sequence of numbers. **Carry part** records the *overflow depth* (defined in section 1.2.) of each flow. It is an atom sketch composed of L_C counters, each of which contains w_2 bits ($2^{w_2} \geqslant d$ is a requisite). We denote this atom sketch with $\mathcal{C}$, and the j^{th} counter with $C[j]$. **Deletion part** is used to support deletions. It is an atom sketch composed of L_D counters, each of which contains w_3 bits. We denote this atom sketch with $\mathcal{D}$, and the j^{th} counter with $D[j]$.

Each atom sketch I_i, $\mathcal{C}$, and $\mathcal{D}$ is associated with k_1, k_2, and k_3 hash functions, whose output is uniformly distributed in the range $[1, L_i]$, $[1, L_C]$, and $[1, L_D]$. And we denote the j^{th} hash function with $h_j^i(.)$, $h_j^C(.)$, and $h_j^D(.)$, respectively.

The **initialization** for the Diamond sketch is simply setting all counters $I_i[j]$, $C[l]$, and $D[m]$ to 0, where $1 \leqslant i \leqslant d$, $1 \leqslant j \leqslant L_i$, $1 \leqslant l \leqslant L_C$, and $1 \leqslant m \leqslant L_D$. Next, we will discuss the insertion, query, and deletion operation.

3.3 Insertion

Algorithm 1: Insertion for Diamond sketch

```
    Input : Item $e$ to insert.
1   Function Insertion($e$):
2       $dep \leftarrow 0$
3       $no\_overflow \leftarrow false$
4       repeat
5           $dep \leftarrow dep + 1$
6           $S_1 \leftarrow \{I_{dep}[h_i^{dep}(e)] \mid 1 \leqslant i \leqslant k_1\}$
7           $V_{min} \leftarrow min(S_1)$
8           if $V_{min} = 2^{w_1} - 1$ then
9               for each counter $c \in S_1$ do
10                  $c \leftarrow 0$
11          else
12              $no\_overflow \leftarrow true$
13              for each counter $c \in S_1$ do
14                  if $c = V_{min}$ then
15                      $c \leftarrow c + 1$
16      until $no\_overflow = true$ or $dep = d$
17      $S_2 \leftarrow \{C[h_j^C(e)] \mid 1 \leqslant j \leqslant k_2\}$
18      for each counter $c' \in S_2$ do
19          if $c'$ is less than $dep - 1$ then
20              $c' \leftarrow dep - 1$
```

As shown in Algorithm 1, given an incoming packet with flow ID e (packet e for short), to increment the flow size, we first update the increment part and then the carry part. We compute the k_1 hash functions of the first atom sketch I_1: $h_1^1(e), h_2^1(e), ..., h_{k_1}^1(e)$, and increment the smallest one(s) of the k_1 counters $I_1[h_1^1(e)], I_1[h_2^1(e)], ..., I_1[h_{k_1}^1(e)]$ (we call them k_1 *mapped counters* for short) by 1. If there are more than one counters that have the smallest value, we would increment them all by 1. If all the k_1 counters hold the value $2^{w_1} - 1$, which is the largest number that w_1 bits can represent, then incrementing them by 1 will result in *overflows*.

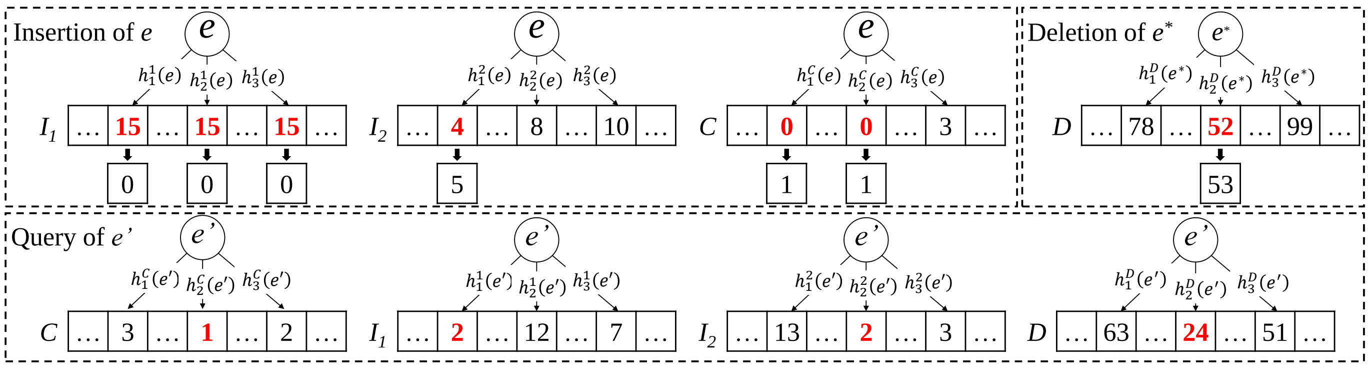


Fig. 2. Insertion, query and deletion of the Diamond sketch.

When the k_1 mapped counters in I_i overflow, the following steps need to be taken: **1)** set the values of all k_1 mapped counters $I_i[h_1^i(e)], I_i[h_2^i(e)], ..., I_i[h_{k_1}^i(e)]$ to 0; **2)** compute the k_1 hash functions of I_{i+1}: $h_1^{i+1}(e), h_2^{i+1}(e), ..., h_{k_1}^{i+1}(e)$, and increment the smallest one(s) of the k_1 counters $I_{i+1}[h_1^{i+1}(e)], I_{i+1}[h_2^{i+1}(e)], ..., I_{i+1}[h_{k_1}^{i+1}(e)]$ by 1; **3)** if all the k_1 mapped counters in I_{i+1} overflow, then repeat step 1 and 2 until the *termination condition* is satisfied, and we record the overflow depth *dep*. There are two termination conditions: there is no overflow in I_i $(1 \leqslant i \leqslant d)$ (we set *dep* to i) or the last atom sketch I_d overflows (we set *dep* to d). **4)** Compute the k_2 hash functions of $\mathcal{C}$, and get the smallest value of the k_2 counters: $C[h_1^{\mathrm{C}}(e)], C[h_2^{\mathrm{C}}(e)], ..., C[h_{k_2}^{\mathrm{C}}(e)]$. If the smallest value is smaller than $dep - 1$, we need to update the carry part to guarantee that the smallest value equals $dep - 1$. To achieve this, for each counter in $C[h_1^{\mathrm{C}}(e)], C[h_2^{\mathrm{C}}(e)], ..., C[h_{k_2}^{\mathrm{C}}(e)]$, if its value is less than $dep - 1$, we set its value to $dep - 1$; otherwise, we do nothing.

To increment the flow size by x (x is larger than 1), we first add x to the mapped counter at the first layer which has the smallest value p. Then we check all the mapped counters again, for each counter with value smaller than $p + x$, we set its value to $p + x$.

3.4 Deletion

Algorithm 2: Deletion for Diamond sketch

```
  Input : Item $e$ to delete.
1 Function Deletion(e):
2     $S \leftarrow \{D[h_j^{\mathrm{D}}(e)] \mid 1 \leqslant j \leqslant k_3\}$
3     $V_{min} \leftarrow min(S)$
4     if $V_{min} \neq 2^{w_3} - 1$ then
5         for each counter $c \in S$ do
6             if $c = V_{min}$ then
7                 $c \leftarrow c + 1$
```

As discussed in Section 1.2, the deletion operation is not requisite in per-flow measurement, but can be added if needed in other scenarios. To delete a packet e, the Diamond sketch only updates its deletion part. We compute the k_3 hash functions of $\mathcal{D}$: $h_1^{\mathrm{D}}(e), h_2^{\mathrm{D}}(e), ..., h_{k_3}^{\mathrm{D}}(e)$, and check the corresponding k_3 counters $D[h_1^{\mathrm{D}}(e)], D[h_2^{\mathrm{D}}(e)], ..., D[h_{k_3}^{\mathrm{D}}(e)]$. If all k_3 mapped counters are $2^{w_3} - 1$, which is the biggest number that w_3 bits can represent, we do nothing; otherwise, we increment the smallest one(s) among the k_3 mapped counters by 1.

3.5 Query

Algorithm 3: Query for Diamond sketch

```
   Input  : Item $e$ to query.
   Output: Query result $V_{query}$.
1  Function Query(e):
2      $S_1 \leftarrow \{C[h_j^{\mathrm{C}}(e)] \mid 1 \leqslant j \leqslant k_2\}$
3      $dep \leftarrow min(S_1) + 1$
4      for $i \in range[1, dep]$ do
5          $S_2 \leftarrow \{I_i[h_j^i(e)] \mid 1 \leqslant j \leqslant k_1\}$
6          $V_i \leftarrow min(S_2)$
7      $V_{insert} \leftarrow \sum_{i=1}^{dep} V_i \cdot (2^{w_1})^{i-1}$
8      $S_3 \leftarrow \{D[h_j^{\mathrm{D}}(e)] \mid 1 \leqslant j \leqslant k_3\}$
9      $V_{delete} \leftarrow min(S_3)$
10     $V_{query} \leftarrow V_{insert} - V_{delete}$
11     return $V_{query}$
```

When querying the size of a flow with ID e, we need to check the status of all the three parts. **First,** we check the carry part. Specifically, the Diamond sketch computes the k_2 hash functions of $\mathcal{C}$ and returns the smallest value among $C[h_1^{\mathrm{C}}(e)], C[h_2^{\mathrm{C}}(e)], ..., C[h_{k_2}^{\mathrm{C}}(e)]$. **Second,** we check the increment part. Specifically, we add 1 to this smallest value returned by the carry part, and suppose the result is *dep*, so we need to check *dep* atom sketches $I_1, I_2, ..., I_{dep}$. For each atom sketch I_i in these *dep* atom sketches, we compute k_1 hash functions and get the smallest value V_i of those k_1 mapped counters $I_i[h_1^i(e)], I_i[h_2^i(e)], ..., I_i[h_{k_1}^i(e)]$. Once all *dep* smallest values have been calculated, the query result determined by increment part and carry part is computed by the following formula: $V_{insert} = \sum_{i=1}^{dep} V_i \cdot (2^{w_1})^{i-1}$. **Third,** we need to check the deletion part. Specifically, we compute the k_3 hash functions of $\mathcal{D}$ and return the smallest value (V_{delete}) among $D[h_1^{\mathrm{D}}(e)], D[h_2^{\mathrm{D}}(e)], ..., D[h_{k_3}^{\mathrm{D}}(e)]$. **Finally,** we compute $V_{insert} - V_{delete}$ as the query result.

3.6 Examples

We will use three examples to illustrate the insertion, query and deletion operations of the Diamond sketch, as illustrated in Figure 2. In those examples, we assume that $d = 4$, $(w_1, w_2, w_3) = (4, 2, 7)$, and $(k_1, k_2, k_3) = (3, 3, 3)$.

To record a packet e, first, we insert it into I_1 in the increment part. Specifically, we compute three hash functions and locate the three counters $I_1[h^1_1(e)], I_1[h^1_2(e)], I_1[h^1_3(e)]$, and find out that all of the three counters have the value 15, so incrementing the smallest counters will cause overflows. We set all the three counters to 0, and insert e into I_2. Second, we compute three hash functions and locate the three corresponding counters $I_2[h^2_1(e)], I_2[h^2_2(e)], I_2[h^2_3(e)]$. They have value 4, 8, 10, respectively, and the smallest value is 4 stored in $I_2[h^2_1(e)]$. We increment it by 1 and $I_2[h^2_3(e)]$ becomes 5. Since there is no overflow in I_2, we record *dep* $= 2$ and update the carry part in the next step. Third, we locate the three counters $C[h^C_1(e)], C[h^C_2(e)], C[h^C_3(e)]$. $C[h^C_1(e)]$ and $C[h^C_2(e)]$ have the value of 0, which is smaller than 1 (computed by *dep* $- 1$), so we need to set $C[h^C_1(e)]$ and $C[h^C_2(e)]$ to 1; $C[h^C_3(e)]$ has the value 3, which is greater than 1, so we do nothing.

Similarly, to query a packet e', first, we check the carry part (by computing three hash functions and locate the three counters $C[h^C_1(e')], C[h^C_2(e')], C[h^C_3(e')]$) and find out that the smallest value is 1, so we need to check I_1 and I_2 to determine the value of e'. Second, we check I_1 and get the smallest value 2, and check I_2 and get the smallest value 2. The query result determined by increment part and carry part is computed as follows: $2 + 2 \cdot 16 = 34$. Third, we check the deletion part. The smallest value of e' in the deletion part is 24, so the final query result is $34 - 24 = 10$.

To delete a packet e^*, we update the deletion part by incrementing the smallest counter, $D[h^D_2(e^*)]$, from 52 to 53.

3.7 Other Uses of the Diamond Sketch

The Diamond sketch can be used to address other important issues in computer networks as well. Here we take *top-k* and *multiple-set membership query* as two examples.

Top-k: The *top-k problem* is to find the k largest elephant flows from the network traffic. A classic method [13] is to use a CM sketch and a min-heap to address this issue. In this paper, we propose to use a Diamond sketch to replace the CM sketch. The process is as follows: **1)** For each incoming packet e, we first insert it, then query it, in the Diamond sketch, and get its size *val*; **2)** We search e in the min-heap, if found, increment its size by 1, and go back to step 1; **3)** If not found, we check the size of the min-heap. If its size is less than k, we insert (e, val) into the min-heap; **4)** Otherwise, we compare *val* with the value of the element in the root node of the min-heap (the one with the smallest size in the heap), and if *val* is larger, we pop out the element at the root, and insert $(e, val + 1)$ into the min-heap; otherwise, we do nothing.

Multiple-set Membership Query: Given sets $S_1, S_2, ..., S_n$, the *multiple-set membership query problem* is to determine which set an item e belongs to. To address this problem, suppose an item e is from set S_i. We regard i as the size of e, and insert e into the Diamond sketch for i times. One classical algorithm for this problem is the BUFFALO [27], which uses one individual Bloom filter for each set.

3.8 Limitations

Since the family of sketch algorithms mainly target the case where limited memory is available, we will not address the problem of maintaining flow identifiers, which consumes a significant amount of memory. Besides, all our experiments are conducted using sketches like CM, CU, CSM, C, and ASketch, all of which do not keep flow identifiers as well.

4 MATHEMATICAL ANALYSES

In this section, we make mathematical analyses on the error rate of the Diamond sketch, and derive the upper bound of the space required to answer a query with a specific error and probability in the presence of skewed distributions.

Since the accurate bound of the CU sketch is really hard to derive [28], [29], in this part, we will mainly focus on the result when the atom sketch is the CM sketch. Assume the skewness of the Zipfian distribution is z. For the situation $z < 1$, which could be named as *moderate skew*, the experimental results in later sections show our algorithms outperform others significantly. As for the situation $z > 1$, we will show in this section that the upper bound of the required space of the Diamond sketch is smaller than that of the CM sketch under the same restriction of error and probability.

4.1 Facts about Zipfian distribution

As pointed by the literature [30], there are two facts as follows. For a Zipfian distribution with parameter z, if we denote the relative frequency of the ith most frequent item as f_i, then $f_i = \frac{c_z}{i^z}$, where c_z is an scaling constant.

Fact 1: For $z > 1$, $1 - \frac{1}{z} \leq c_z \leq z - 1$.

Fact 2: For $z > 1$, $\frac{c_z k^{1-z}}{z-1} \leq \sum_{i=k}^{U} f_i \leq \frac{c_z (k-1)^{1-z}}{z-1}$, where U is the range of the distribution.

4.2 Upper bound of the CM sketch

From [30] we could know for a Zipfian distribution with parameter z, the upper bound of required space for a CM sketch to answer a query with error $\epsilon||\alpha||$ with probability at least $1 - \delta$ is $O(\epsilon^{-\min\{1,1/z\}} \ln \frac{1}{\delta})$, where α is the size of the flow.

4.3 Upper bound of Diamond sketch

First, given parameters (ϵ, δ), set the width of the carry part $w_c = \frac{e}{\epsilon}$ and the number of hash functions $k_d = \ln \frac{1}{\delta}$, then with probability at least $1 - \delta$, $\hat{a}_i \leq a_i + ||\alpha|| k_d^{1-z}/w$, where $\hat{a}_i$ is the result of estimation of the ith item. Since the query result of the carry part is quite small and always a integer, if we adjust the size of the width and the number of hash functions, we could guarantee that it returns the right answer with a probability $1 - \delta$ and the cost is rather small and calculable. Also, this fact that the error rate of the carry part is consistent could be partially illustrated by Fig. 16.

To better illustrate the problem, we will mainly focus on the situation when the Diamond sketch has two layers. When the Diamond sketch has only one layer, it is exactly the same as CM sketch and it doesn't need the help of the carry part. As for the situation of more that two layers, the analyses are basically the same, but would be much more complicated in terms of form, so we would only discuss the situation of two layers here.

The error could be divided into two parts. The first is the collisions with larger items and the second is the collisions with smaller items. By choosing hash functions and settings like $k = \frac{w}{3}$, we could avoid the collisions with kth larger items with constant probability ($\frac{2}{3}$).

In the first layer, when the result of the carry part is correct, suppose the estimation is $\hat{a}_i$,

$$\begin{aligned}
\Pr[\hat{a_i} > a_i + \epsilon||\alpha||] &= \Pr[\forall_j \text{count}[h_j(i)] > a_i + \epsilon||\alpha||] \\
&= \Pr[\forall_j X_{i,j} > a_i + \epsilon||\alpha||] \\
&= \Pr[\forall_j a_i + \frac{1}{w}\sum_{n=k+1,x}^{U} a_x > a_i + \epsilon||\alpha||] \\
&< \Pr[\frac{1}{w}\sum_{n=k+1,x}^{U} a_x > \epsilon||\alpha||] \\
&< \frac{2}{3}^d
\end{aligned}$$

The third line is derived from the fact mentioned before, and the fifth line the Markov inequality.The correct rate hence could be guaranteed by the choice of parameters.

In the deeper layers, the number of "small flows" decreases significantly (simply by applying the beformentioned facts again), which means we could use lesser counters to balance the memory usage of the carry part. As long as the depth of the increment part is enough, the memory usage of Diamond sketch would surly be lesser than that of the CM sketch under the same restriction in terms of error and probability.

5 Performance Evaluation

5.1 Metrics

The performance of sketches is often evaluated by accuracy and speed with a fixed size of memory. The accuracy is usually measured by *AAE* and *ARE*, and speed is usually measured by *Throughput*.

Average Absolute Error (AAE): Let f_e be the real size of a packet e, and $\hat{f}_e$ be the estimated size of the flow e reported by the sketch, then the *absolute error (AE)* of flow e is calculated as $|f_e - \hat{f}_e|$, and the average absolute error is calculated as AAE $= \frac{1}{|S|}\sum_{e \in S}|f_e - \hat{f}_e|$, where S is the query set.

Average Relative Error (ARE): The *relative error (RE)* of a packet e is calculated as $|f_e - \hat{f}_e|/f_e$, and the average relative error is calculated as ARE $= \frac{1}{|S|}\sum_{e \in S}\frac{|f_e - \hat{f}_e|}{f_e}$.

Throughput: Let n be the number of certain operations (insertion, deletion or query) executed on a sketch, and t be the total execution time in nanosecond. The throughput can be calculated as Throughput $= \frac{1000 \cdot n}{t}$Mops, where *Mops* is the abbreviation of *Million Operations per Second*.

5.2 Experimental Setup

5.2.1 Datasets

To extensively evaluate the performance of Diamond, we use three kinds of datasets as the input: real IP trace streams, a real-life transactional dataset, and synthetic datasets. The methods we use to acquire those datasets and generate the query set are discussed below.

5.2.1.1 Real IP trace streams: The real IP trace streams are captured by the main gateway of our campus, which is a collection of flows. We adopt the most widely used method that considers the source IP address and the destination IP address as the flow ID. In the IP trace streams, there are totally $12,285,667$ entries, including $763,206$ distinctive flows, whose sizes vary from 1 to $56,889$, with a mean of 16.1, and a variance of $4,729$. Notice that 52.9% flows have the size of 1, so when the query result of a flow is $1,001$, whose real size is 1, the relative error will be 1000 for this flow. Therefore, the experimental results of average relative error seem large, but the error is already small enough. We have compared our real IP trace with CAIDA trace, they are similar in flow size distributions: about half of flows have only one packet, and the average flow size is $3 \sim 10$.

5.2.1.2 Real-life transactional dataset: This dataset is generated from a spidered collection of web HTML documents. First, each document is filtered; second, a set of all the distinct item (which is regarded as a flow) is extracted from the document; finally, a distinct transaction is generated based on that set. This dataset can be downloaded from [31]. Since the whole dataset is considerably large, we choose 10% consecutive entries from the original dataset as our experimental dataset, which has totally $32,475,964$ entries, including $980,759$ distinct items (flows). The numbers of occurrences of items vary from 1 to $65,327$, with a mean of 33.1, and a variance of $14,218.6$.

5.2.1.3 Synthetic datasets: Our synthetic datasets are generated by a performance testing tool called Web Polygraph [32]. There are 11 generated datasets, which all follow the Zipfian [11] distribution with the skewness ranging from 0.0 to 1.0. Note that when the skewness is 0.0, the distribution is actually uniform. Each dataset has exactly 10M strings, and each distinct string represents a flow ID, taking up 13 bytes to represent the five-tuple. The maximum size of flow increases from 28 to $21,423$ as the skewness increases from 0.0 to 1.0.

5.2.1.4 Query Set: In our experiments, the query set is exclusively composed of all the items belonging to the dataset, and each item only appears once in the query set. Notice that in the recent paper of the Augmented sketch [10], the query set is generated differently: the top-k items/flows are queried a great number of times, while other flows are not queried. The details of the query set are not available from the paper, so we cannot reproduce the experimental results of the Augment sketch paper. Using our query set, our following experiments show that it has almost the same accuracy as the CM sketch.

5.2.2 Parameters Setting

We compare our Diamond sketch with five typical sketches: CM sketches [13], CU sketches [17], Count (C) sketches

[18], CSM sketches [19], and Augmented (A) sketches [10]. For those five kinds of sketches, each sketch has 4 hash functions, and every counter has 16 bits, a proper size to accommodate the maximal size of flows appearing in the three kinds of datasets described above. We set the filter size of the Augmented sketch to be 32, as recommended by the authors. We allocate 0.5 MB memory for each of the five sketches and the Diamond sketch.

5.2.3 *Implementation*

We use C++ as the programming language and implement the insertion, query, and deletion (if possible) operation for C, CU, CM, CSM, Augmented (based on CM sketches), and Diamond sketches. In our code, we implement the data structure single-threaded using a single core. The hash functions we use are BOB Hash acquired from an open source website [33]. We have made the source code of our experiments available on GitHub [1].

5.2.4 *Computation Platform*

We performed all the experiments on a machine with 4-core CPUs (8 threads, Intel Core i7@2.5 GHz) and 16 GB total DRAM memory. CPU has three levels of cache memory: two 32KB (where 1KB = 2^{10} bytes) L1 caches for each core, one 256KB L2 cache for each core, and one 6MB (where 1MB = 2^{20} bytes) L3 cache shared by all cores.

5.3 Parameters of Diamond Sketch

To maximize the performance of Diamond, we choose appropriate parameters for Diamond: **1)** To fully utilize every bit in the carry part, d (number of atom sketches in the increment part) needs to be the power of 2, and we use $d = 4$ in our experiments. **2)** According to our tests, to achieve high accuracy, the ratio p of memory usage of the carry part to that of the increment part should be in range $[0.1, 0.2]$, and we choose $p = 0.15$ in our experiments. **3)** According to our experimental results of various datasets, when $\{L_1, L_2, ..., L_d\}$ is a geometric sequence with a *common ratio* (L_2/L_1) between 4.0 and 5.0, the performance of Diamond is high without obvious fluctuations. Therefore, we choose the median value 4.5 for the common ratio.

5.4 Accuracy

In this section, we present the experimental results on AAE and ARE of Diamond compared to C, CU, CSM, CM, and A sketches. We use *Dia* to represent our Diamond sketch in experimental figures due to space limitation.

1) Experiments on AAE

Absolute Error CDF on three types of datasets: *Our experimental results, reported in Figure 3, 4, and 5, show that the AAE is similar on the three types of datasets, which are 1) synthetic dataset with a skewness of 1.0, 2) real IP trace streams, and 3) real-life transactional dataset, respectively.* Therefore, in the following experiments, we will focus on the experimental results on synthetic datasets and real IP trace streams.

We cut off because the y-axis value of the Diamond sketch has become the maximum value – 1. Although the accuracies of other sketches may be not that bad on skewed traffic distributions, the experiments show clearly that out

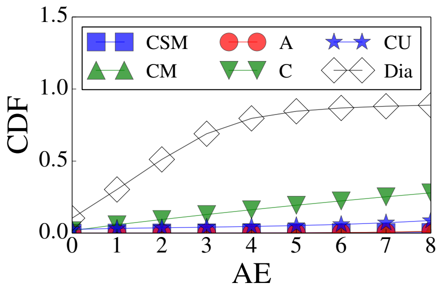


Fig. 3. Absolute Error CDF on Transactional Dataset.

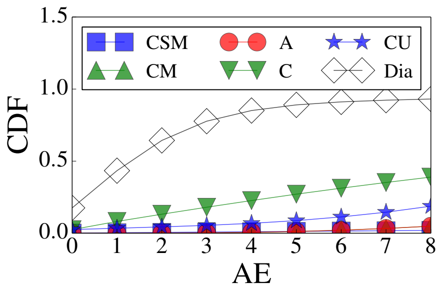


Fig. 4. Absolute Error CDF on Real IP Traces.

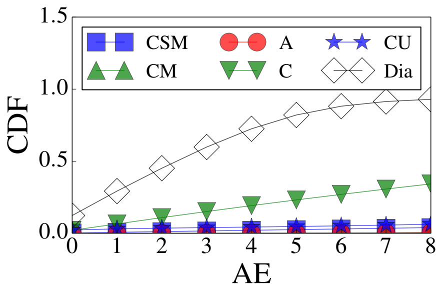


Fig. 5. Absolute Error CDF on Synthetic Dataset with a Skewness of 1.0.

Diamond sketch does much better than all of them. At the tail of the distributions, the y-axis value of all algorithms approaches the maximum value – 1. This figure shows that when the memory is tight, Diamond is the best choice to achieve smallest error. When memory size is large, user can choose any sketch to achieve almost zero error.

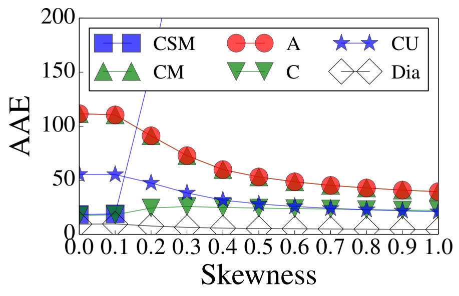


Fig. 6. Absolute Error vs. Skewness.

Absolute Error vs. Skewness: *Our experimental results, reported in Figure 6, show that as the skewness of the synthetic dataset ranges from 0.0 to 1.0, the average absolute error of Diamond sketch is* $[1.9, 308.4]$, $[8.7, 11.8]$, $[8.7, 11.8]$, $[1.9, 4.9]$,

$[4.6, 6.2]$ *times smaller than the average absolute errors of CSM, CM, A, C, CU sketches, respectively.* The average absolute error of our Diamond sketch drops from 9.6 to 4.5 as the skewness of the synthetic dataset increases from 0.0 to 1.0.

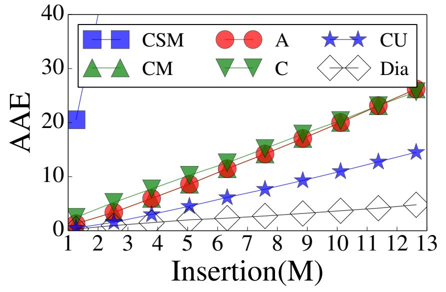


Fig. 7. Absolute Error vs. # of Insertions.

Absolute Error vs. # of Insertions: *Our experimental results, reported in Figure 7, show that as the number of the insertions of the real IP trace ranges from 1.3M to 12.7M, the average absolute error of Diamond sketch is* $[56.1, 196.0]$, $[3.6, 5.5]$, $[3.6, 5.5]$, $[5.1, 7.6]$, $[1.6, 3.1]$ *times smaller than the average absolute errors of CSM, CM, A, C, CU sketches, respectively.* Initially, we insert the first 10% of the real IP traces into the sketches, and the number of insertions is 1.3M. We then calculate the average absolute errors of the sketches. Following the same process, we keep on inserting the next 10% of the IP traces into the sketches, calculating the corresponding average absolute errors, until all IP traces are inserted. AAE of the Diamond sketch is better than that of the CU sketch even when the number of insertions of Diamond sketch is twice over that of CU sketch.

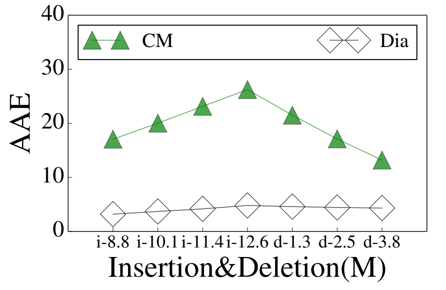


Fig. 8. Absolute Error vs. # of Deletions.

Absolute Error vs. # of Deletions: *Our experimental results, reported in Figure 8, show that as the number of the deletions of the real IP trace ranges from 1.3M to 3.8M, the average absolute error of Diamond sketch is* $[3.1, 4.7]$ *times smaller than the average absolute error of CM sketches.* We do not show the deletion results of other sketches, as they either do not support deletion or do not show the deletion algorithm in their papers. To measure the absolute errors of sketches during deletions, we first insert all of the real IP traces into the sketches, and then measure the average absolute errors after deleting the last 10%, 20%, 30% of the IP traces from the sketches. When when making N random deletions, the result is almost the same.

Absolute Error vs. Memory Size: *Our experimental results, reported in Figure 9, show that, in case of real IP trace streams, as the overall memory size ranges from 0.25 MB to 2.0 MB, the average absolute error of Diamond sketch is* $[26.6, 50.3]$, $[2.3, 6.0]$, $[2.4, 6.0]$, $[4.7, 5.4]$, $[1.4, 3.4]$ *times smaller than the average absolute errors of CSM, CM, A, C, CU sketches, respectively.* Our Diamond sketch outperforms other sketches more observably in terms of AAE when the memory size is limited, such as 0.5 MB.

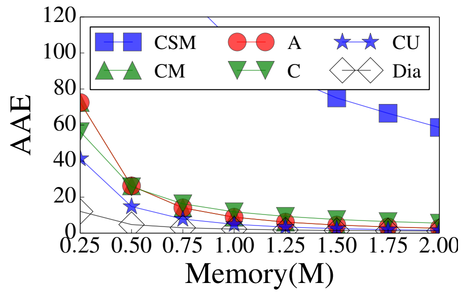


Fig. 9. Absolute Error vs. Memory Size.

Indeed, when the memory size is large enough, all sketches can achieve almost zero error, which makes the comparisons meaningless. However, we argue that for the high speed network traffic, the sketch should be stored in the on-chip memory. The on-chip memory is often ten or more times faster than the off-chip memory (*e.g.*, 16GB DRAM) [7], [34]. However, the on-chip memory (*i.e.*, SRAM, often less than several MBs) is very expensive and limited in size. In such cases, to catch up with the line speed, the Diamond sketch will be the best choice rather than other sketches.

2) Experiments on ARE

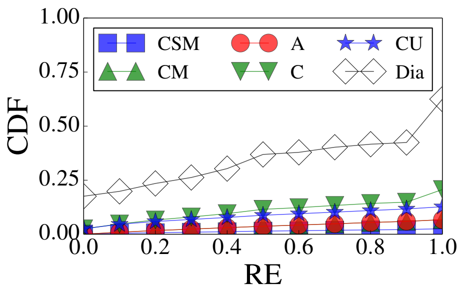


Fig. 10. Relative Error CDF.

Relative Error CDF: *In Figure 10, when the x-axis value is 1.0, the y-axis value means the percentage of flows whose relative error is less than 1.0. Our experimental results, reported in Figure 10, show that in case of real IP trace streams, the percentage of Diamond is 62.46%, which is 24.6, 9.29, 9.29, 3.02, and 4.91 times higher than the corresponding percentages for CSM, CM, A, C, CU sketches, respectively.*

Relative Error vs. Skewness: *Our experimental results, reported in Figure 11, show that as the skewness of the synthetic dataset ranges from 0.0 to 1.0, the average relative error of Diamond sketch is* $[2.0, 383.3]$, $[10.9, 12.6]$, $[10.9, 12.6]$, $[1.9, 6.1]$, $[6.0, 7.2]$ *times smaller than the average relative errors of CSM, CM, A, C, CU sketches, respectively.* The Diamond sketch has a steady high performance no matter how the skewness of the dataset changes (uniform or non-uniform).

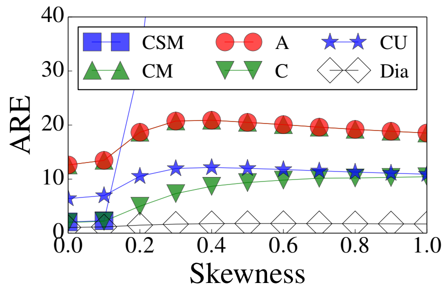


Fig. 11. Relative Error vs. Skewness.

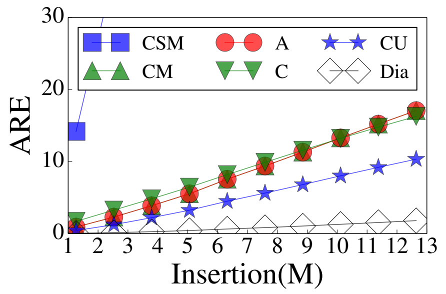


Fig. 12. Relative Error vs. # of Insertions.

Relative Error vs. # of Insertions: *Our experimental results, reported in Figure 12, show that as the number of the insertions of the real IP trace increases from 1.3M to 12.6M, the average relative error of Diamond sketch is* $[255.8, 342.3]$, $[9.5, 18.2]$, $[9.5, 18.2]$, $[9.0, 34.0]$, $[5.7, 9.0]$ *times smaller than the average relative errors of CSM, CM, A, C, CU sketches, respectively.*

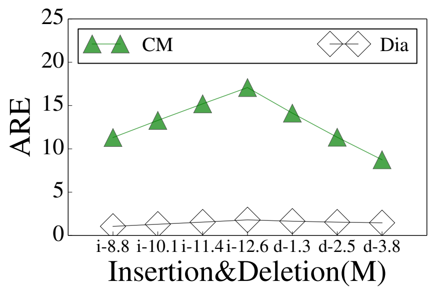


Fig. 13. Relative Error vs. # of Deletions.

Relative Error vs. # of Deletions: *Our experimental results, reported in Figure 13, show that as the number of the deletions of the real IP trace ranges from 1.3M to 3.8M, the average relative error of Diamond sketch is* $[6.0, 8.6]$ *times smaller than the average relative error of CM sketches. Figure 14 shows that the sequence of deletions has limited influence on accuracy, compared to Figure 13, in which deletions are completed sequentially.*

Relative Error vs. Memory Size: *Our experimental results, reported in Figure 15, show that as the overall memory size ranges from 0.25 MB to 2.0 MB, the average relative error of Diamond sketch is* $[34.0, 508.3]$, $[7.7, 21.7]$, $[7.7, 21.7]$, $[5.9, 39.8]$, $[4.6, 11.6]$ *times smaller than the average relative errors of CSM, CM, A, C, CU sketches, respectively.* Our Diamond sketch outperforms other sketches more observably in terms of ARE when the memory size is limited.

Absolute Error CDF on Real IP Traces: *Our experimental results, reported in Figure 16, show that the AAE of the Diamond sketch is always smaller when the atom sketch of the increment part is the CU sketch rather than the CM sketch. The change of the atom sketch of the carry part makes very small difference in terms of accuracy, because for the carry part, it can achieve very high accuracy as it only needs to record the overflow times of large flows while the number of large flows is small. Thus the results are not included.*

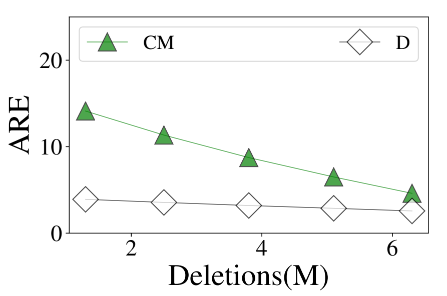


Fig. 14. Relative Error vs. # of Deletions.

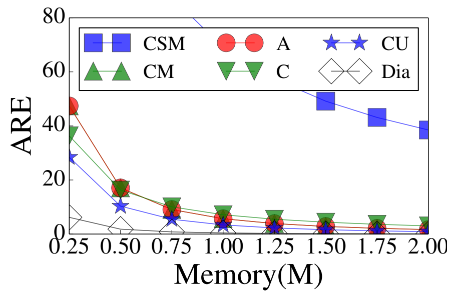


Fig. 15. Relative Error vs. Memory Size.

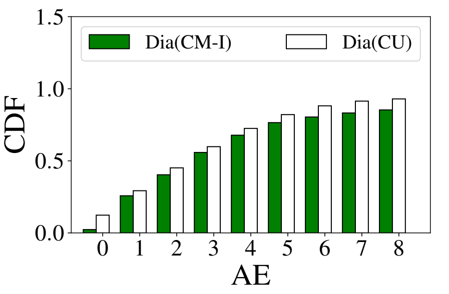


Fig. 16. Absolute Error CDF on Real IP Traces.

5.5 Speed

In this section, we present the experimental results on *# of memory accesses* and *throughput* of Diamond compared to C, CU, CSM, CM, and A sketches. We use software simulations to measure the number of memory accesses. Specifically, we use a variable to record the number of memory accesses. When one counter is accessed, we increment the corresponding variable.

# of (Query) Memory Accesses vs. # of Insertions: *Our experimental results, reported in Figure 17, show that as the number of the insertions of the real IP trace ranges from 1.3M to 12.7M, the number of query memory accesses of Diamond sketch is* $[6.3, 6.7]$, *which is comparable to CSM, CM, A, C, CU*

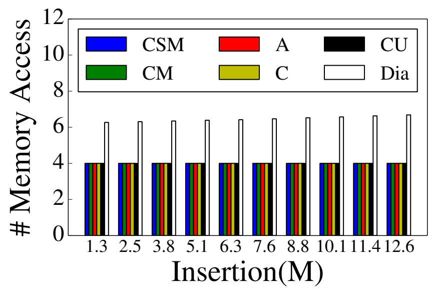


Fig. 17. # of (Query) Memory Accesses vs. # of Insertions.

sketches. As discussed in Section 3, when querying a flow e, our Diamond sketch needs to check both the carry part and the increment part, thus needs more memory accesses than the prior art.

Although our Diamond sketch needs more memory accesses, the throughput is still high owing to better cache behavior. Specifically, the memory size of $\mathcal{C}$ and I_1 is much smaller than other sketches and each query will access these two parts, so they will be cached with high probability. Therefore, the query speed of the Diamond Sketch is comparable to other sketches.

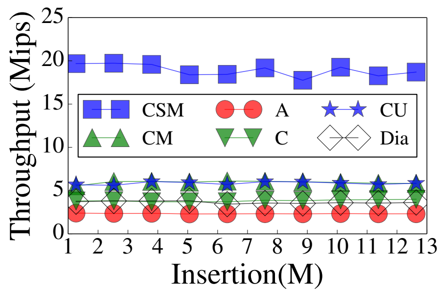


Fig. 18. Insertion Throughput vs. # of Insertions.

Insertion Throughput vs. # of Insertions: *Our experimental results, reported in Figure 18, show that as the number of the insertions of the real IP trace ranges from 1.3M to 12.7M, the throughput of our Diamond sketch during insertions is* $[3.4, 3.8]$ *Mops, which is comparable to CSM, CM, A, C, CU sketches.* CSM does only one memory access during insertion, so its insertion throughput is higher than other sketches. **Query Throughput vs. # of Insertions:** *Our experimental results, reported in Figure 19, show that as the number of the insertions of the real IP trace ranges from 1.3M to 12.7M, the throughput of our Diamond sketch during queries is* $[3.0, 3.5]$ *Mops, which is comparable to CSM, CM, A, C, CU sketches.* **Deletion Throughput vs. # of Insertions:** *Our experimental results, reported in Figure 20, show that as the number of the insertions of the real IP trace ranges from 1.3M to 12.7M, the throughput of our Diamond sketch during deletions is* $[1.4, 1.7]$ *times higher than CM sketch.*

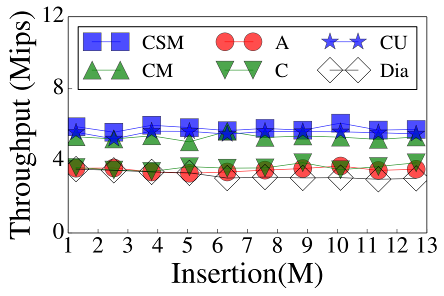


Fig. 19. Query Throughput vs. # of Insertions.

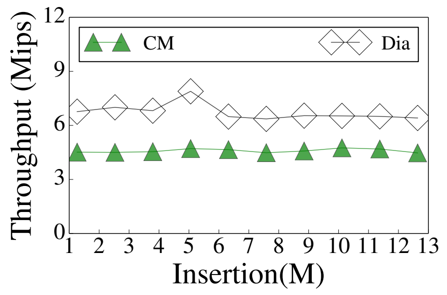


Fig. 20. Deletion Throughput vs. # of Insertions.

5.6 Experiments on Other Uses of Diamond

The metrics and experimental setup of the two sets of experiments are as follows. As for the top-k problem, let S_1 be the set of all the real top-k flows, and S_2 be the set of all the estimated top-k flows, then *the correct rate of the top-k problem* is defined as $\frac{|S_1 \cap S_2|}{|S_1|}$, and *the AAE of the top-k problem* is determined by the AAE of $S_1 \cap S_2$. As for the multiple-set membership query problem, *the correct rate of the multiple-set membership query problem* represents the proportion of items whose query result is correct. We use synthetic datasets for both problems. The multiple-set data are generated such that the sizes of sets conform to the Zipfian distribution with the skewness of 1.0. As for the top-k problem, we keep the maximum size of the min-heap to k, and as for the multiple-set membership query problem, half of the queried items belong to a certain set, and the other half do not belong to any one of the multiple sets. The Count sketch [18], the Space Saving [35] and BUFFALO [27] are the most widely used algorithms in top-k and multiple-set membership query, respectively. Therefore, we compare Diamond with them in our experiments. The Diamond sketch greatly outperforms the prior art in experimental results because of its adaptation to skewed datasets.

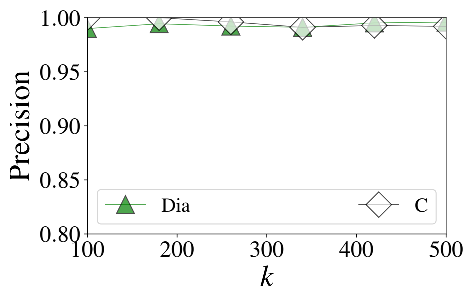


Fig. 21. (Top-k) Precision vs. k.

(Top-k) Precision vs. k: *Our experimental results, reported in Figure 21, show that as k ranges from 100 to 500, the Precision of our Diamond sketch is comparable to that of the Count sketch.*

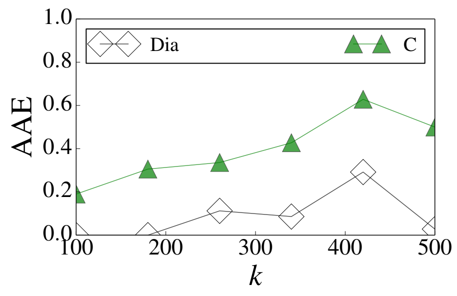


Fig. 22. (Top-k) AAE vs. k.

(Top-k) AAE of Correct Flows vs. k: *Our experimental results, reported in Figure 22, show that as k ranges from 100 to 500, the AAE of our Diamond sketch is* $[2.15, 300]$ *times lower than that of the Count sketch.*

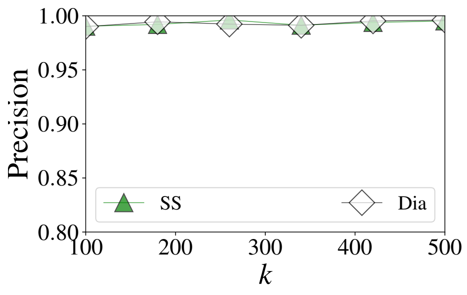


Fig. 23. (Top-k) Precision vs. k.

reviewA(Top-k) Precision vs. k: *Our experimental results, reported in Figure 23, show that as k ranges from 100 to 500, the precision of our Diamond sketch is comparable to that of the Space-Saving.*

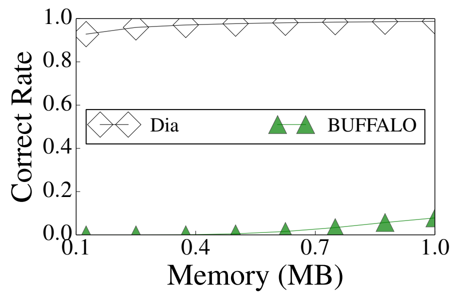


Fig. 24. (Multiple-set) Correct rate vs. Memory Size.

(Multiple-set) Correct Rate vs. Memory: *Our experimental results, reported in Figure 24, show that as memory usage ranges from 100 to 500, the correct rate of our Diamond sketch is* $[12.6, 1940]$ *times higher than that of the BUFFALO.*

(Multiple-set) Throughput vs. Dataset: *Our experimental results, reported in Figure 25, show that for all the four datasets, the throughput of our Diamond sketch is* $[10.2, 11.1]$ *times higher than that of the BUFFALO.*

6 Conclusion

Per-flow measurement is a criticial issue in computer networks, and sketch is a popular data structure used to address this problem. Existing sketches have poor memory efficiency and are not accurate enough, especially for non-uniform datasets. In this paper, we propose the Diamond sketch, including increment part, carry part, and deletion part. Our Diamond sketch can dynamically assign an appropriate number of atom sketches to record the size of each flow in an on-demand way, thus can optimize memory efficiency. We compared our Diamond sketch with five typical sketches: CM, CU, Count, Augmented, and CSM sketches, and extensive experimental results show that the Diamond sketch outperforms the best of the five typical sketches by up to two orders of magnitude in terms of relative error. To guarantee that our experimental results are reproducible and objective, we release the source code of all the six sketches on GitHub [1].

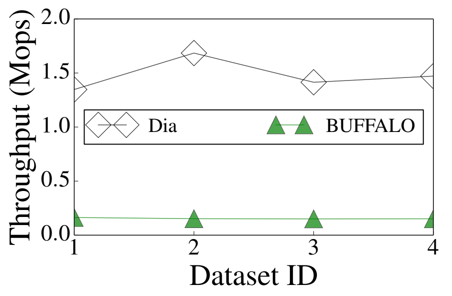


Fig. 25. (Multiple-set) Throughput on multiple datasets.

7 Acknowledgment

We would like to thank the anonymous reviewers for their insightful suggestions. This work is supported by Primary Research Development Plan of China (2018YFB1004403, 2016YFB1000304), NSFC (61672061). Yulong Shen (ylshen@mail.xidian.edu.cn) is the corresponding author.

References

[1] S. C. of Diamond sketch and related sketches, " https://github.com/data-kth/Diamond-Sketch.git."

[2] X. Dimitropoulos, P. Hurley, and A. Kind, "Probabilistic lossy counting: an efficient algorithm for finding heavy hitters," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 1, pp. 5–5, 2008.

[3] C. Estan and G. Varghese, "New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice," *ACM Transactions on Computer Systems (TOCS)*, vol. 21, no. 3, pp. 270–313, 2003.

[4] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen, "Sketch-based change detection: methods, evaluation, and applications," in *Proc. ACM IMC*, pp. 234–247.

[5] X. Li, F. Bian, and et al., "Detection and identification of network anomalies using sketch subspaces," in *Proc. ACM SIGCOMM*, 2006, pp. 147–152.

[6] M. Yoon, T. Li, S. Chen, and J.-K. Peir, "Fit a spread estimator in small memory," in *Proc. IEEE INFOCOM 2009*.

[7] Y. Li, R. Miao, C. Kim, and M. Yu, "Flowradar: a better netflow for data centers," in *Proc. USENIX NSDI*, 2016.

[8] K. Cheng, L. Xiang, and M. Iwaihara, "Time-decaying bloom filters for data streams with skewed distributions," in *Proc. IEEE RIDE-SDMA 2005*, pp. 63–69.

[9] G. Cormode, "Sketch techniques for approximate query processing," *Synposes for Approximate Query Processing: Samples, Histograms, Wavelets and Sketches, Foundations and Trends in Databases. NOW publishers*, 2011.

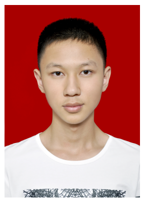
[10] P. Roy, A. Khan, and G. Alonso, "Augmented sketch: Faster and more accurate stream processing," in *Proc. ACM SIGMOD*, 2016, pp. 1449–1463.

[11] D. M. Powers, "Applications and explanations of zipf's law," in *Proceedings of the joint conferences on new methods in language processing and computational natural language learning*. Association for Computational Linguistics, 1998, pp. 151–160.
[12] L. A. Adamic and B. A. Huberman, "Power-law distribution of the world wide web," *science*, vol. 287, no. 5461, pp. 2115–2115, 2000.
[13] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.
[14] C. Baquero, P. S. Almeida, R. Menezes, and P. Jesus, "Extrema propagation: Fast distributed estimation of sums and network sizes," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 4, pp. 668–675, 2012.
[15] T. Buddhika, M. Malensek, S. L. Pallickara, and S. Pallickara, "Synopsis: A distributed sketch over voluminous spatiotemporal observational streams," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 11, pp. 2552–2566, 2017.
[16] D. Tong and V. K. Prasanna, "Sketch acceleration on fpga and its applications in network anomaly detection," *IEEE Transactions on Parallel and Distributed Systems*, 2017.
[17] C. Estan and G. Varghese, "New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice," *ACM Transactions on Computer Systems (TOCS)*, vol. 21, no. 3, pp. 270–313, 2003.
[18] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," in *International Colloquium on Automata, Languages, and Programming*. Springer, 2002, pp. 693–703.
[19] T. Li, S. Chen, and Y. Ling, "Per-flow traffic measurement through randomized counter sharing," *IEEE/ACM Transactions on Networking (TON)*, vol. 20, no. 5, pp. 1622–1634, 2012.
[20] W. Feng and H. Mounir, "Matching the speed gap between sram and dram," in *Proc. IEEE HSPR*, 2008, pp. 104–109.
[21] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.
[22] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: a scalable wide-area web cache sharing protocol," *IEEE/ACM TON*, vol. 8, no. 3, pp. 281–293, 2000.
[23] S. Cohen and Y. Matias, "Spectral bloom filters," in *Proc. ACM SIGMOD*, 2003, pp. 241–252.
[24] J. Aguilar-Saborit, P. Trancoso, V. Muntes-Mulero, and J.-L. Larriba-Pey, "Dynamic count filters," *ACM SIGMOD Record*, vol. 35, no. 1, pp. 26–32, 2006.
[25] Y. Lu, A. Montanari, B. Prabhakar, S. Dharmapurikar, and A. Kabbani, "Counter braids: a novel counter architecture for per-flow measurement," *Proc. ACM SIGMETRICS*, pp. 121–132, 2008.
[26] S. Ramabhadran and G. Varghese, "Efficient implementation of a statistics counter architecture," *SIGMETRICS Perform. Eval. Rev.*, vol. 31, no. 1, pp. 261–271, Jun. 2003. [Online]. Available: http://doi.acm.org/10.1145/885651.781060
[27] M. Yu, A. Fabrikant, and J. Rexford, "Buffalo: Bloom filter forwarding architecture for large organizations," in *Proc. ACM Conex*, 2009, pp. 313–324.
[28] G. Einziger and R. Friedman, "A formal analysis of conservative update based approximate counting," in *2015 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 2015, pp. 255–259.
[29] G. Bianchi, K. Duffy, D. Leith, and V. Shneer, "Modeling conservative updates in multi-hash approximate count sketches," in *2012 24th International Teletraffic Congress (ITC 24)*. IEEE, 2012, pp. 1–8.
[30] G. Cormode and S. Muthukrishnan, "Summarizing and mining skewed data streams," in *Proceedings of the 2005 SIAM International Conference on Data Mining*. SIAM, 2005, pp. 44–55.
[31] R. life transactional datasets, "http://fimi.ua.ac.be/data/."
[32] A. Rousskov and D. Wessels, "High-performance benchmarking with web polygraph," *Software: Practice and Experience*, 2004.
[33] H. website, "http://burtleburtle.net/bob/hash/evahash.html."
[34] T. Yang, G. Xie, Y. Li, Q. Fu, A. X. Liu, Q. Li, and L. Mathy, "Guarantee ip lookup performance with fib explosion," in *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4. ACM, 2014, pp. 39–50.
[35] A. Metwally, D. Agrawal, and A. El Abbadi, "Efficient computation of frequent and top-k elements in data streams," in *International Conference on Database Theory*. Springer, 2005, pp. 398–412.

Tong Yang received his Ph.D. degree in Computer Science from Tsinghua University in 2013. Now he is an associate professor in Computer Science Department, Peking University, China. His research interests include data streams, sketches, measurements, and machine learning. He published papers in SIGCOMM, SIGKDD, SIGMOD, SIGCOMM CCR, VLDB, ATC, ToN, JSAC, ICDE, INFOCOM, ICNP, ICDCS, etc.

Siang Gao is a senior student and an undergraduate researcher at the Institute of Network Computing and Information Systems (NS&IS), School of EECS, Peking University. His research interests include network traffic measurement and big data analysis.

Zhouyi Sun is a senior student and an undergraduate researcher at the Institute of Network Computing and Information Systems (NS&IS), School of EECS, Peking University. His research interests include network traffic measurement and big data analysis.



Yufei Wang is a senior student and an undergraduate researcher at the Institute of Network Computing and Information Systems (NS&IS), School of EECS, Peking University. His research interests include network traffic measurement and big data analysis.

Yulong Shen received the B.S. and M.S. degrees in computer science and the Ph.D. degree in cryptography from Xidian University, Xian, China, in 2002, 2005, and 2008, respectively. He is currently a Professor with the School of Computer Science and Technology, Xidian University, and also an Associate Director of the Shaanxi Key Laboratory of Network and System Security. He has also served on the technical program committees of several international conferences, including the ICEBE, the INCoS, the CIS, and the SOWN. His research interests include wireless network security and cloud computing security.

Xiaoming Li is a professor in computer science and technology and the director of Institute of Network Computing and Information Systems (NCIS) at Peking University, China. His current research interest is in search engine and web mining. He led the effort of developing a Chinese search engine (Tianwang) since 1999, and is the founder of the Chinese web archive (Web InfoMall).