Randomized Error Removal for Online Spread Estimation in High-Speed Networks

Haibo Wang, *Graduate Student Member, IEEE*, Chaoyi Ma, *Graduate Student Member, IEEE*, Olufemi O. Odegbile, Shigang Chen, *Fellow, IEEE*, and Jih-Kwon Peir

***Abstract*— Flow spread measurement provides fundamental statistics that can help network operators better understand flow characteristics and traffic patterns with applications in traffic engineering, cybersecurity and quality of service. Past decades have witnessed tremendous performance improvement for single-flow spread estimation. However, when dealing with numerous flows in a packet stream, it remains a significant challenge to measure per-flow spread accurately while reducing memory footprint. The goal of this paper is to introduce new multi-flow spread estimation designs that incur much smaller processing overhead and query overhead than the state of the art, yet achieves significant accuracy improvement in spread estimation. We formally analyze the performance of these new designs. We implement them in both hardware and software, and use real-world data traces to evaluate their performance in comparison with the state of the art. The experimental results show that our best sketch significantly improves over the best existing work in terms of estimation accuracy, packet processing throughput, and online query throughput.**

***Index Terms*— Traffic measurement, flow spread, randomization, online, sketches.**

I. Introduction

TRAFFIC measurement in high-speed networks has many challenging problems. One of them is how to efficiently and accurately estimate flow spread over a packet steam during a measurement period. A packet steam is sequence of packets, each modelled as $\langle f, e\rangle$, where f is flow label and e is element identifier. Both f and e can be defined arbitrarily based on information from packet header or payload for specific application need. Flow spread is defined as the number of distinct elements carried by each flow in the packet stream. It provides fundamental statistics that can help network operators better understand flow characteristics and traffic patterns with applications in traffic engineering, cybersecurity and quality of service [2], [3], [4], [5], [6].

There are many practical applications that can benefit from flow spread estimation, including P2P hot-spot localization [7], web caching prioritization [8], [9], detection of DDoS attacks [10], [11], port scanning measurement [2] and worm propagation detection [12], [13]. Below we give several examples in the context of Internet applications. Consider a packet stream that arrives at a high-speed router. For example, if we consider all packets from the same source address as a flow and use destination addresses as the elements under monitoring, a flow-spread measurement module deployed at a gateway router will detect potential external adversaries that are scanning the internal network — these are external sources with large spreads (i.e., their flows contain too many distinct destinations), or in case of stealthy scanning they are sources with modest spreads at any measurement period but persisting at a spread level higher than normal over a long time [14]. As an opposite example, if we use destination addresses as flow labels and source addresses as elements, spread measurement will help identify the victims of possible DDoS attacks — these are internal destination addresses with large spreads (i.e., their flows contain too many distinct source addresses). In yet another example, a large server farm may learn the popularity of its content by tracking the number of distinct users accessing each file [15], where all users accessing the same file form a flow. Finally, spread measurement has also been applied in various data analysis systems at Google [16]. For instance, Sawzall [17], Dremel [18] and PowerDrill [19] estimate the number of distinct users that search the same key, where we can model all search requests for the same key as a flow and user identities (e.g., their IP addresses) as the elements in each flow.

This paper is interested in per-flow spread measurement, allowing users to query the spread of any flow online in real time. We have three performance requirements for the design of a spread measurement module. First, it should incur low processing overhead per packet in order to support high-rate streaming. Second, it should be memory-efficient in order to support software/hardware implementation on the data plane of a streaming device which may operate on cache memory. Third, it should support efficient online spread queries to support real-time applications. We again use packet stream in high-speed networks as example to justify the requirements. Modern routers forward packets at hundreds of gigabits or

Manuscript received 21 April 2021; revised 28 September 2021 and 15 April 2022; accepted 21 July 2022; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor S. Schmid. This work was supported by NSF under Grant SCC-2124858, Grant CSR-1909077, and Grant NeTS-1719222. A preliminary version of this paper has been published in the Proceedings of the VLDB Endowment, Copenhagen, Denmark, August 16-20, 2021 [DOI: 10.14778/3447689.3447707]. *(Corresponding author: Shigang Chen.)*

Haibo Wang, Chaoyi Ma, Shigang Chen, and Jih-Kwon Peir are with the Department of Computer and Information Science and Technology, University of Florida, Gainesville, FL 32611 USA (e-mail: wanghaibo@ufl.edu; ch.ma@ufl.edu; sgchen@cise.ufl.edu; peir@cise.ufl.edu).

Olufemi O. Odegbile is with the Department of Computer Science, Clark University, Worcester, MA 01610 USA (e-mail: oodegbile@clarku.edu).

This article has supplementary downloadable material available at https://doi.org/10.1109/TNET.2022.3197968, provided by the authors.

Digital Object Identifier 10.1109/TNET.2022.3197968

even terabits per second (at least 8.3M packets or 83M packets per second considering maximum transmission unit for Ethernet is 1500 bytes). Tracking a large number of flows simultaneously can be a serious challenge. Specifically, if one wants to perform online flow spread measurement in real time, one way is to implement the measurement module on data-plane network processors. Since their on-chip circuitry and cache memory have to be shared among many other routing/performance/security functions, low overhead and memory efficiency become highly desirable properties in order not to create performance bottleneck.

There are two categories of solutions for per-flow spread measurement. One category estimates each flow with a separate data structure, called *spread estimator*. To count distinct elements, it must be able to remember the elements that it has seen. Such single-flow estimators include bitmap [20], FM [21], multi-resolution bit-map [3], SMB [22], KMV [23], LogLog [24] and HyperLogLog (HLL) [16], [25]. They require hundreds or thousands of bits for each flow in order to achieve good accuracy and range. When the number of flows is numerous, monitoring all flows with separate data structures can be too costly. We need more compact data structures called *spread sketches* that monitor all flows simultaneously without linearly increasing the memory overhead, which leads to the second category of solutions [9], [26], [27], [28]. They share a certain number of spread estimators among all flows when recording their elements. But sharing causes error in spread estimation. When a flow shares an estimator with other flows, the estimator produces the combined spread of all those flows instead of the spread of an individual flow. To reduce the error, the current approach [9], [26], [27] follows the idea of CountMin [29] by mapping each flow to multiple estimators, making multiple spread estimations, and taking the minimum answer. However, it is well known that this approach has a positively biased error that can be very large when the multiple estimators are all shared with other large flows. Moreover, since each flow has to be recorded in multiple estimators and each query has to be computed from multiple estimators, both the processing overhead per packet and the query overhead for each flow are increased multi-fold.

The goal of this paper is to design new spread sketches for online per-flow spread estimation (in the scenarios described before) that incur much smaller processing overhead and query overhead than the start of the art, yet result in much less error in spread estimation. More specifically, we want the processing overhead and the query overhead to be multiple times smaller, and the error to be an order of magnitude smaller. To achieve these seemingly conflicting objectives, we cannot follow the prior approaches but need to explore new paths toward compact and efficient recording of packets in a way that enables error removal. We introduce two new sketch designs, called randomized error-reduction sketch (rSkt) and unit-level randomized error-reduction sketch (rSkt2). Their basic idea is to spread the error due to estimator sharing evenly between a primary estimator and a complement estimator, so that the error can be subtracted away. Moreover, we also present an improved design of rSkt, denoted as rSkt1, to reduce the query overhead. We formally analyze the performance of these new sketches. We implement them in both hardware and software, and use real-world data traces to evaluate their performance in comparison with the state of the art. The experimental results show that our best sketch significantly improves over the best existing work in terms of estimation accuracy (up to 99.5% estimation error reduction), packet processing throughput (up to 126% throughput improvement), and online query throughput (up to 3 times throughput improvement), thanks to its randomized error-reduction design.

II. Background and Prior Art

A. Problem Statement

Consider the problem of monitoring the packet stream received by a router or other network devices. Packets form a *flow* if they carry the same *flow label* f, which can be protocol identifier, source address, destination address, port numbers, and/or other fields in the packet headers, depending on the type of measurement functions and application requirements. The packet stream under monitoring may consist of numerous flows whose packets interleave arbitrarily. Within a flow, each packet is abstracted as an element e, which may again be a header-field combination or content in packet payload.

Flow spread is defined as the number of distinct elements carried by a flow f. It is worth noting that flow spread is quite different from *flow size* [30], [31], [32], [33], [34], which is defined as the number of elements in a flow. As an example, consider a packet stream $\{\langle f_1, e_1\rangle, \langle f_2, e_2\rangle, \langle f_1, e_1\rangle, \langle f_1, e_1\rangle\}$, the size of flow f_1 is 3 as packet $\langle f_1, e_1\rangle$ appears three times while the spread of flow f_1 is 1 as the second and fourth packets are duplicates of the first packet $\langle f_1, e_1\rangle$. *Single-flow spread estimation* refers to estimating the spread for a single flow, which is more difficult than single-flow size measurement. In contrast, the problem of *multi-flow spread estimation* measures the spreads of numerous flows simultaneously given any flow label of interest, which is this paper will focus on. We provide the problem statement: *design an efficient sketch (i.e., compact data structure) that records the packets of a given stream, and support online queries for spread estimates on any given flow labels. Online queries are performed live when we process the packet stream. They are important for applications that require real-time responses. In contrast, offline queries are performed after the packet stream has been processed [4], [6], [9], [28], and thus not subject to any real-time requirement.*

For a continuous packet stream, measurement is typically done in each pre-defined period, and the content of the sketch is offloaded to a server for long-term storage after each period.

For a concrete example of zero delay cybersecurity, Internet worms scan the Internet to identify and infect vulnerable hosts. As more and more hosts are infected, they join the effort of scanning, causing an exponential infection curve before leveling off. A modern worm can infect the Internet in a matter of minutes. The infected hosts are super spreaders as they scan. We will prefer to identify them in real time instead of after each epoch (which may be in minutes).

In some applications, we only need to monitor the super spreaders (those with very large spreads) [26], [35].

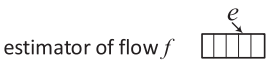


Fig. 1. An estimator is an array of units (bits, FM registers or HLL registers). Any element e of flow f will be recorded in one of the units.

However, there are other scenarios where the spread information of non-super spreaders is also useful. For example, to avoid detection, stealthy scanners may probe a small number of destination addresses/ports at any time but do so persistently over a long period. If we measure the spreads of all flows and analyze such information over time, we will be able to find these stealthy scanners that are not aggressive at any instant but persist in low-rate scanning. In another example, suppose that an intrusion detection system identifies a set of worm-infected hosts that perform probing to infect others. With per-flow measurement, we will be able to examine these hosts in the measurements taken from the previous periods and find out when each of them begins its probing (which results in spread increase). This helps us to establish infection timeline among the hosts for trace-back purpose. Moreover, we can query their current probing rates in real time as per-flow measurement is performing in the current period. In general, measuring the spreads of all flows enables us to perform broad analysis over long-term flow behaviors in order to detect subtle anomalies that deviate from the norm. Additional applications of per-flow measurement on stealthy attack detection, fine-grained traffic analysis, flow loss map, ECMP debugging, and TCP timely attack detection can be found in [31].

It is more difficult to measure the spread of a flow than doing so for the size of a flow (which requires only a counter). The reason is that we have to "remember" the elements that have been seen before in order to remove duplicate elements, and that takes a lot of memory space if the flow has a very large number of distinct elements. The overhead can be greatly reduced if we provide a spread estimate. In this paper, we refer to a data structure that records the elements of a flow and provides a spread estimate as an *estimator*. Below we discuss the related work, beginning with single-flow spread estimators.

B. Single-Flow Spread Estimators

To monitor the spread of a flow, a naive solution is to use a hash table to store the received elements for duplicate removal [36], [37], but this is costly as a flow may have millions or billions of distinct elements.

More efficient single-flow spread estimators include bitmap [3], [20], [26], FM sketch [21], and HLL sketch [16], [25]. We unify their description as follows: As shown in Figure 1, each estimator is an array U of m units, where each unit, $U[i]$, $0 \leq i < m$, is a bit, a 32-bit register, or a 5-bit register for bitmap, FM or HLL, respectively. The memory consumption of single-flow spread estimators are shown in Table I.

When receiving an element e of the flow, we hash e to one of the units, $U[h(e)]$, for recording, where $h(.)$ is a hash function. The recording operation depends on the unit type. In case of bitmap, we set $U[h(e)]$ to one. In case of FM, we choose the $G(e)$th bit in $U[h(e)]$ to set, with $G(e) = i, 0 \leq i < 32$ with probability $(\frac{1}{2})^{i+1}$, where $0 \leq i < 32$. In case of HLL, we update $U[h(e)] = \max\{U[h(e)], G(e) + 1\}$.

TABLE I
MEMORY NEED FOR DIFFERENT SINGLE-FLOW SPREAD ESTIMATORS WITH m UNITS

Estimators	Memory	Remark
bitmap	m	$m \ln m > n$. n is the real spread of flow
FM	$32m$	recommended as $m = 128$
HLL	$5m$	recommended as $m = 128$

When we estimate the flow's spread, we average across the array. This computation also depends on the unit type. In case of bitmap [20],

$$avg = \frac{\sum_{i=0}^{m-1} U[i]}{m}.$$

In case of FM [21], let $\rho(U[i])$ be the number of consecutive ones starting from the lowest-order bit in $U[i]$.

$$avg = \frac{\sum_{i=0}^{m-1} 2^{\rho(U[i])}}{m}$$

In case of HLL [16], [25], harmonic averaging is used to tame the impact of outliers.

$$avg = \frac{m}{\sum_{i=0}^{m-1} \frac{1}{2^{U[i]}}}$$

From the average, we can estimate the flow spread based on the formulas from the papers cited above. For example, in case of bitmap [3], [20], the spread is estimated as $-m \ln(1-avg)$.

While UnivMon [30] and ElasticSketch [33] are designed to measure the sizes of flows, they also estimate the number of flows, called *cardinality*, in a packet stream. For this purpose, they treat the whole stream as a single giant flow and the flow labels as elements. They belong to single-flow spread estimators, and their memory overhead is very large when comparing with bitmap/FM/ HLL.

C. Multiflow Overhead Challenge

To monitor multiple flows, we may assign each flow a separate spread estimator. With 5,000 bits, a bitmap estimator has an estimation range up to just $-5000 \times \ln(1/5000) = 42,586$, according to [20]. To achieve good accuracy, an FM (HLL) estimator will need hundreds or thousands of bits [28]. For example, when an HLL estimator takes 640 bits when it uses 128 registers of 5 bits each [9]. Consider one million concurrent flows. The memory requirement for one million bitmaps is 5000 Mb, while that for one million HLL estimators is 640 Mb, which can be a serious problem for online operations, particularly when its implementation uses on-chip cache memory for high speed [9], [38], [39], such as on a network processor for Internet traffic. Today's switches may have 128MB SRAM [40], but this cache memory has to store the routing table and support essential routing/security/performance functions. Moreover, there may be multiple measurement tasks. Therefore, it is highly desirable to minimize the memory consumption of any measurement task.

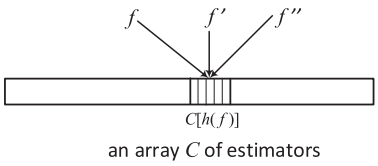


Fig. 2. Flow f is hashed to estimator $C[h(f)]$, which carries error from other flows due to hash collision. Recall that each estimator is an array of units.

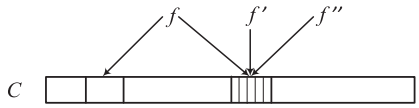


Fig. 3. Hashing each flow f to d estimators, where $d > 1$. Note that f' and f'' are also hashed to d estimators each, which are not drawn.

To save space, if we use fewer estimators than the number of flows, each estimator will have to handle multiple flows. For example, we may hash the flows to an array C of estimators, as is shown in Figure 2, where flows f, f' and f'' are hashed to the same estimator. When we query for the spread of flow f, the estimator will produces an estimate that carries noise (error) from f' and f'' due to hash collision.

We give an example to show the error can be very large in practical scenarios. Suppose that the allocated memory is 10Mb and the number of flows is 10^6, which is validated by the fact that a 10-min CAIDA dataset contains 2.52M flows if we consider each source-destination pair as a flow. Considering the most compact single-flow spread estimator, i.e., HLL, which occupies 640 bits under recommended setting, if each packet is recorded in one estimator, on average, 64 flows share one estimator. Therefore, the error can be very large. Below we explain how the existing literature handles such error.

D. Existing Multiflow Estimators

There are two approaches to reduce error caused by hash collision. One is to hash each flow f to d estimators, as shown in Figure 3, where $d = 2$. The d estimators each produce a spread estimation for flow f. The smallest of the d estimations carries the least error. Essentially, this approach [26], [27] uses the CountMin idea [29] but replaces counters with spread estimators. This idea has been proposed by [43] and a generalized design called bSkt [9] that uses different single-flow spread estimators, including bitmap, FM, and HLL, as plug-ins are implemented and evaluated.

For online spread queries, the above approach has two problems. First, even though error is reduced by taking the smallest, our experiments still show significant error. Second, the query computation overhead increases d-fold because, for each flow, we have to perform estimation from d estimators instead of one. This is fine if it is done offline, but will be a problem if it is done online as we process the live stream. Note that in order to ensure decent error reduction, d cannot be too small. Our goal is to design a new sketch that reduces error to a level much lower than bSkt [9] and in the meantime reduces query computation as well.

Another approach in the literature for reducing estimation error is virtual sketches [4], [6], [9], [28], which share a large array of units (e.g., bits, FM/HLL registers) for all flows. More specifically, they construct virtual estimators for individual flows from these shared units. Each flow has its own virtual estimator, which produces a spread estimation that carries error from other flows due to unit sharing. Removing this error will require memory access to the whole unit array and computation across the whole unit array. Such overhead is many times larger than that of bSkt [9], making it unsuitable for online spread queries. Specifically, in our test, the online query throughput of bSkt is at least 50-300 times larger than that of virtual sketches. We will not consider these approach further in the paper for the desire of supporting online queries. We compare existing work and ours in Table II.

TABLE II
COMPARISON AMONG EXISTING MULTI-FLOW ESTIMATORS AND OUR WORK

Estimators	Accuracy	Recording overhead	Query overhead
bSkt [9]	Low	Medium	Medium
cSkt [9]	Low	Medium	Medium
OpenSketch [27]	Low	Medium	Medium
CSE [4]	High	Low	High
vHLL [28]	High	Low	High
vSkt [9]	High	Low	High
AROMA [41], [42]	High	Low	High
Randomized Sketches	High	Low	Low

Marple [44] is a query language that is backed by a new programmable key-value store primitive on switch hardware. It evicts the flow's information to DRAM when the on-chip cache is saturated and answers the online queries only using the information stored in SRAM. BeauCoup [45] focuses on detecting super spreaders for a number of applications (which has different types of flow labels) simultaneously with one-time update. This is different from the problem this paper studies, which is to measure spreads for all flows under a single application (which has one type of flow label).

The work in [36] studies super spreader detection. This paper differs from them as our focus is *per-flow* spread estimation. Per-flow spread estimation requires a sketch to record the elements of all flows while the work in [36] stores a limited number of flow-element pairs or flow labels and samples out most small (sometimes medium) flows. It is suitable for super-spreader measurements, but does not keep any information for numerous flows that are sampled out. This is particularly true when there exist very large flows whose pairs will push out most other flows, according to our experiments and paper [46].

AROMA [41] and [42] is the state of the art on spread estimation. It employs the idea of sampling and provides the flexibility on the accuracy-memory tradeoff. With more memory allocated, more flows will be tracked and the accuracy will be improved. We will compare with AROMA in the evaluation and show that our work advances in some metrics.

The error removal techniques are proposed in [38] and [43], where the authors propose eliminating the estimation bias in flow size estimation using the CountMin sketch. Specifically, by selecting one counter in each of the d rows and taking the minimum among the values in the d counters, we can

TABLE III
NOTATIONS

$C, \bar{C}$	hash table of estimators
$V(.)$	result of an estimator
f, e	flow label and element identifier
$s_f / \hat{s}_f$	actual/estimated spread of f
d	No. of hashed estimators per packet
w	No. of estimators per hash table
m	No. of units per estimator
$h(.) \in [0, w)$	uniform hash function
$g(.) \in \{0, 1\}$	uniform hash function
$g'(.,.) \in \{0, 1\}$	two-input uniform hash function

simulate querying the size of a flow that does not exist and hence take this value as the noise and remove it from the estimate. This technique can also be used in CountMin-based spread estimation solutions. The work in [47] proposes that we only need to maintain the value of the total spread in the packet stream, from which we can easily calculate the noise and then subtract it away.

III. RANDOMIZED ERROR-REDUCTION SKETCHES

In this section, we introduce two new randomized sketches for flow spread measurement that produce spread estimates with much lower error and much lower computation time than the prior art.

A. Hash Table

Let's first revisit the hash table approach in Section II-C and Figure 2. The hash table is an array C of w estimators. The ith estimator in the table is denoted as $C[i]$, $0 \leq i < w$. Each estimator is an array of m units which may be bits, FM registers or HLL registers as explained in Section II-B and Figure 1. The jth unit in the ith estimator is denoted as $C[i][j]$, $0 \leq i < w$, $0 \leq j < m$.

• *Recording*: Consider an arbitrary flow f. It is hashed to $C[h(f)]$. After we receive an packet $\langle f, e \rangle$, we hash the element e to a unit, $C[h(f)][h(e)]$, where it is recorded based on the unit type according to Section II-B. Note that a modulo operation is always assumed in this paper to keep the hash output in the proper range. For example, $C[h(f)][h(e)]$ should actually be $C[h(f) \mod w][h(e) \mod m]$. Some important notations are shown in Table III for reference.

• *Querying*: Upon receiving a spread query on flow f, we produce an estimate from $C[h(f)]$ based on its type (bitmap, FM or HLL); see Section II-B. The result is denoted as $V(C[h(f)])$, where $V(.)$ refers to the type-dependent estimation formula.

Let F_f be the set of flows that are hashed to $C[h(f)]$, i.e., $\forall f' \in F_f, h(f') = h(f) \mod w$. Let s_f be the true spread of flow f. The number of distinct packets $\langle f, e \rangle$ that are actually recorded by $C[h(f)]$ is $\sum_{f' \in F_f} s_{f'}$, which is greater than or equal to s_f since $f \in F_f$. The noise with respect to flow f is $\sum_{f' \in F_f - \{f\}} s_{f'}$, which is what we want to remove.

B. Baseline Randomized Error-Reduction Sketch - rSkt

Our first solution, called randomized error-reduction sketch (rSkt), is to use two hash tables, C and $\bar{C}$, each of w spread estimators, as shown in Figure 4, where each flow f is hashed to a pair of candidate estimators, $C[h(f)]$ and $\bar{C}[h(f)]$. Let $g(.)$ be a function that maps f to 0 or 1 pseudo-randomly with equal probability. All elements of flow f will be recorded in $C[h(f)]$ if $g(f) = 0$ or in $\bar{C}[h(f)]$ if $g(f) = 1$. We call the estimator that records elements of f as the flow's *primary estimator* and the other as the flow's *complement estimator*. In practice, we may implement $g(f)$ by taking the least-order bit of hash value $h'(f)$ using another uniform hash function $h'(.)$ or taking the highest-order bit of $h(f)$ before modulo w.

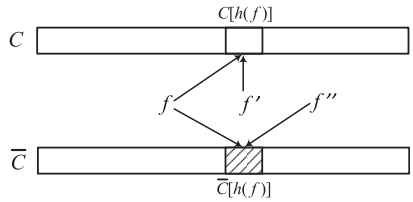


Fig. 4. Flow f is hashed to a primary estimator ($\bar{C}[h(f)]$ for this example) and a complement estimator ($C[h(f)]$ for this example). The noise in $F_f - \{f\}$ is split between these two estimators with equal probability. For example, f' is recorded in $C[h(f)]$ and f'' is recorded in $\bar{C}[h(f)]$.

Consider an arbitrary noise flow $f' \in F_f - \{f\}$, where F_f is the set of flows that $\forall f' \in F_f$, $h(f') = h(f) \mod w$. It is also either recorded in $C[h(f)]$ or $\bar{C}[h(f)]$, with equal probability, depending on the value of $g(f')$. Hence, it is either recorded in flow f's primary estimator or its complement estimator, with equal probability. Therefore, our solution splits error $F_f - \{f\}$ between f's primary estimator and its complement estimator, with equal probability. Roughly speaking, the flow's primary estimator records s_f and about half of the error, and its complement records about half of the error, allowing us to subtract error away. The flow's estimated spread, denoted as $\hat{s}_f$, is computed as follows.

$$\hat{s}_f = (1 - 2g(f))[V(C[h(f)]) - (V(\bar{C}[h(f)])] \quad (1)$$

We stress that the splitting operation of $F_f - \{f\}$ between $C[h(f)]$ and $\bar{C}[h(f)]$ will not be perfect and residual error will remain after subtraction. Due to the pseudo-randomness of hashing, $F_f - \{f\}$ may happen to contain a large flow f' that dominates in $F_f - \{f\}$. Even if the flows in $F_f - \{f\}$ are evenly distributed between $C[h(f)]$ and $\bar{C}[h(f)]$, the error caused by these flows is not evenly distributed between the two. Most error will go where f' goes. In this case, subtraction will not serve its purpose.

One approach to solve the above problem is to use d independent hash functions, $h_i(f)$, $0 \leq i < d$, each mapping f to a pair of candidate estimators, $C[h_i(f)]$ and $\bar{C}[h_i(f)]$. We also use d independent pseudo-random functions, $g_i(f)$, $0 \leq i < d$, each choosing a primary estimator from two candidates: For $0 \leq i < d$, if $g_i(f) = 0$, all elements of f will be recorded in $C[h_i(f)]$; if $g_i(f) = 1$, all elements of f will be recorded in $\bar{C}[h_i(f)]$. Hence, for each received packet $\langle f, e \rangle$, it will be recorded for d times, once in each of f's primary estimators.

To estimate the spread of flow f, we first find the pair of candidate estimators, $C[h_x(f)]$ and $\bar{C}[h_x(f)]$, that has the

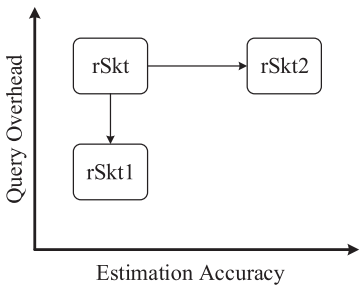


Fig. 5. Design logic for randomized sketches.

smallest combined estimation, i.e.,

$$\exists x \in [0,d), \quad V(C[h_x(f)]) + V(\bar{C}[h_x(f)]) \\ = \min\{V(C[h_i(f)]) + V(\bar{C}[h_i(f)]), 0 \le i < d\}. \quad (2)$$

This is the pair that carries the least combined error in $F_f - \{f\}$ and is thus less likely to contain large flows. Finally, we estimate $\hat{s}_f$ as follows:

$$\hat{s}_f = (1 - 2g(f))[V(C[h_x(f)]) - (V(\bar{C}[h_x(f)])] \quad (3)$$

We find through experiments that by choosing $d > 1$, the estimation accuracy may be improved (in case of using HLL estimators) or may be worse (in case of using bitmap/FM estimators). The reason is that although using the pair with the smallest combined error helps avoid large flows, each element is now recorded for d times in C and $\bar{C}$, which boost overall error among all estimators. Whether estimation accuracy becomes better or worse will depend on the joint impact of the above two factors.

In the following, we will introduce two more sophisticated randomized sketches, i.e., rSkt1 and rSkt2, each improving the performance in one metric. rSkt1 reduces query overhead tremendously, while maintaining the same estimation accuracy and similar recording overhead compared to rSkt, with less than 5% additional memory consumption. rSkt2 improves estimation accuracy for all bitmap/FM/HLL types to a level that rSkt/rSkt1 cannot achieve with any d value. The design logic of randomized sketches is illustrated in Figure 5. Based on application needs (fast query or precise estimation), users can choose to use rSkt1 or rSkt2.

C. Supporting Fast Query With rSkt1

It is desirable that when online recording the high-speed packet stream, the sketch module can support fast query at the same time, such that any spread anomaly, e.g., DDoS attacks, super spreaders, etc., can be detected instantly and handled by the system admin as soon as possible. This requires that the query overhead of sketches should be as low as possible, allowing even per-packet online query without slowing down packet-recording speed. We admit that per-packet online query is demanding in computing resources, but reducing the query overhead will definitely benefit the query process, either making online queries more frequently or doing so with less computing overhead. However, our experimental results on rSkt's throughput demonstrate that its query overhead is at least 10 times higher than its recording overhead, regardless of the types of the estimators we use, making it impractical to answer spread query of the flow label carried by the incoming packet on the fly when recording high-speed packet streams. This motivates the design of rSkt1 that reduces the query overhead to the same level as the recording overhead.

We use the case of using bitmap estimators as an example to show why rSkt's query overhead is much larger than its recording overhead. From Section II-B, each bitmap can be represented as an array U of m bits. rSkt records any packet $\langle f, e\rangle$ to the bitmap by setting the bit $U[h(f|e)]$ to one, where $|$ is the concatenation operator. It answers the spread query by scanning all the m bits in the bitmap, increasing the memory access overhead by m times.

Our solution is that we maintain an additional integer for each bitmap estimator, which is initialized as 0. The integers themselves can form two independent integer arrays, denoted as B and $\bar{B}$, with integer $B[j]$ responsible for $C[j]$ and integer $\bar{B}[j]$ responsible for $\bar{C}[j]$, and allowing parallel processing of recording and query. Every time we set a bit in the bitmap from zero to one, we increase the corresponding integer by one. When answering the query, according to the [20], the spread estimate of any estimator, e.g, $C[j]$, can be calculated as follows:

$$V(C[j]) = -m\ln(1 - B[j]/m), 0 \le j < w \quad (4)$$

The above equation indicates that we only need to access an integer $B[j]$ for any bitmap estimator $C[j]$. In contrast, the querying operation in Section II-B needs to scan all the m bit in $C[j]$, m times of memory access overhead.

To estimate the spread of flow f, we calculate the values of estimators $C[h_i(f)]$ and $\bar{C}[h_i(f)]$, $0 \le i < d$ by (4). After that, following the processing similar to rSkt, we will find the pair of candidate estimators $C[h_x(f)]$ and $\bar{C}[h_x(f)]$, that has the smallest combined estimation, and further estimate the spread of f by (3).

rSkt1 can also support FM and HLL estimators. For FM, we still maintain two integer arrays, B and $\bar{B}$. Recall from Section II-B that an FM estimator is an array U of m registers, each with 32 bits. When recording any packet $\langle f, e\rangle$ to any FM estimator $C[j]$, we choose the $G(f|e)$th bit in register $C[j][h(e)]$ to set, where $G(.)$ is a hash function and outputs $i, 0 \le i < 32$ with probability $(\frac{1}{2})^{i+1}$. The integer $B[j]$ is increased by

$$\rho(C[j][h(e)]) - \rho(C'[j][h(e)]), \quad (5)$$

where $\rho(C[j][h(e)])$ is the number of consecutive ones starting from the lowest-order bit in $C[j][h(e)]$ and $C'[j][h(e)]$ represents the register $C[j][h(e)]$ before update.

When answering the query, according to [21], the spread estimate of estimator $C[j]$ in (4) for bitmap now becomes

$$V(C[j]) = m \cdot 2^{B[j]/m}/\phi, 0 \le j < w \quad (6)$$

where ϕ is a pre-computed constant which is approximately 0.78 when m is large enough.

When using HLL estimators, two integer arrays B and $\bar{B}$ now become two float arrays. From Section II-B, each HLL is an array U of m registers, each with 5 bits. When recording

any packet $\langle f, e\rangle$ to a HLL estimator $C[j]$, we update the register $C[j][h(e)] = \max\{C[j][h(e), G(f|e) + 1\}$, where $G(\cdot) = i, 0 \le i < 31$ with probability $(\frac{1}{2})^{i+1}$. The float $B[j]$ is increased by

$$2^{-C[j][h(e)]} - 2^{-C'[j][h(e)]} \tag{7}$$

where $C'[j][h(e)]$ represents the register $U[i]$ before update.

When answering the query, according to [16], [25], the spread estimate of estimator $C[j]$ in (4) for bitmap now becomes

$$V(C[j]) = \alpha_m \cdot m^2/B[j], 0 \le j < w \tag{8}$$

where α_m is a constant whose value is related to the value of m. When $m \ge 128$, $\alpha_m = 0.7213/(1 + 1.079/m)$.

The recording and querying operations of rSkt1 are given in Algorithms 1 and 3, respectively.

Algorithm 1 Recording a Packet in rSkt1

Input: packet $\langle f, e\rangle$
Action: record e of f in C, $\bar{C}$ and do update in B, $\bar{B}$
for i = 0 to $d - 1$ **do**
  if $g_i(f) = 0$ **then**
    $rSkt1\_Record\_Plugin(f, e, C[j], B[j])$
  else
    $rSkt1\_Record\_Plugin(f, e, \bar{C}[j], \bar{B}[j])$
  end if
end for

Algorithm 2 $rSkt1\_Record\_Plugin(f, e, C[j], B[j])$

Input: packet $\langle f, e\rangle$, estimator $C[j]$ and integer/float $B[j]$
Action: record $\langle f, e\rangle$ in $C[j]$ and update $B[j]$
switch (type of estimator $C[h_i(f)]$)
case bitmap**:**
  if $C[j][h(f|e)]$ = 0 **then**
    $C[j][h(f|e)]$ = 1
    $B[j]$ += 1
  end if
case FM**:**
  $B[j]$ -= $\rho(C[j][h(e)])$
  $C[j][h(e)][G(f|e)]$ = 1
  $B[j]$ +=$\rho(C[j][h(e)])$
case HLL**:**
  $B[j]$ -= $2^{-\rho(C[j][h(e)])}$
  $C[j][h(e)] = \max\{C[j][h(e)], G(f|e) + 1\}$
  $B[j]$ +=$2^{-\rho(C[j][h(e)])}$
end switch

rSkt1 reduces query overhead by maintaining an additional integer/float for each estimator, which accelerates query speed. The cost is additional memory consumption, which we claim is less than 5%. In case of using bitmap estimators, the bitmap contains m bits and the additional integer may take up $\lceil \log_2(m) \rceil$ bits. As a result, we need $\lceil \log_2(m) \rceil / m$ additional memory. Since m is usually very large, e.g., 5000, to accommodate large flows, we have $\lceil \log_2(m) \rceil / m = 0.3\%$. In case of using FM estimators, each FM estimator needs $32m$ bits and m is recommended as 128 [21]. The integer stores the sum of the number of leading zeros in each register, which takes up $\lceil \log_2(32m) \rceil = 12$ bits, resulting in the memory addition of $\frac{12}{32m} = 0.3\%$. In case of using HLL estimators, a float takes up 32 bits, and each HLL estimator needs $5m$ bits where m is recommended as 128 [21], resulting in the memory addition of $\frac{32}{5m} = \frac{1}{128} = 5\%$. Therefore, we can conclude that the memory addition memory consumption can be neglected compared to the memory consumption by rSkt.

Algorithm 3 Querying on a Flow in rSkt1

Input: flow label f, maximum integer value MAX_VALUE
Output: spread estimate
X=MAX_VALUE
for i = 0 to $d - 1$ **do**
  Obtain $V(C[h_i(f)])$ and $V(\bar{C}[h_i(f)])$ by (4) for bitmap, (6) for FM and (8) for HLL.
  if $X > V(C[h_i(f)]) + V(\bar{C}[h_i(f)])$ **then**
    $X = V(C[h_i(f)]) + V(\bar{C}[h_i(f)])$
    $x = i$
  end if
end for
if $g_x(f) = 0$ **then**
  return $V(C[h_x(f)]) - V(\bar{C}[h_x(f)])$
else
  return $V(\bar{C}[h_x(f)]) - V(C[h_x(f)])$
end if

It is worth noting that compared to rSkt, rSkt1 will not make any changes on the estimation accuracy, under the same values of w and d. Next, we will present a new design that significant improves the estimation accuracy compared to rSkt and rSkt1.

D. Unit-Level Randomized Error-Reduction Sketch - rSkt2

We use an example to illustrate the idea behind our third design, referred to as rSkt2. Consider the baseline sketch rSkt with $d = 1$. A flow f is hashed to $C[h(f)]$ and $\bar{C}[h(f)]$. Without loss of generality, suppose that $g(f) = 0$ and f is recorded in $C[h(f)]$. Suppose there are only two flows, f and f', in F_f. Flow f' is a flow of large spread. There are two possible cases.

Case 1: f' is recorded in $C[h(f)]$. Because all elements of f and f' are recorded in $C[h(f)]$, $V(C[h(f)])$ is an estimate of the combined spread of f and f'. Because no element is recorded in $\bar{C}[h(f)]$, $V(\bar{C}[h(f)]) = 0$. Hence, the estimate by (1) becomes $\hat{s}_f = V(C[h(f)]) - V(\bar{C}[h(f)]) = V(C[h(f)])$, which carries large positive error introduced by f'.

Case 2: f' is recorded in $\bar{C}[h(f)]$. Because all elements of f are recorded in $C[h(f)]$, $V(C[h(f)])$ is an estimate of the spread of f. Because all elements of f' are recorded in $\bar{C}[h(f)]$, $V(\bar{C}[h(f)])$ is an estimate of the spread of f'. As $\hat{s}_f = V(C[h(f)]) - V(\bar{C}[h(f)])$, it is the estimated spread of f minus the estimated spread of f', thus carrying large negative error introduced by f'.

To resolve the above dilemma, we have to look deeper at $C[h(f)]$ and $\bar{C}[h(f)]$ into their unit-level structures and break up f' into pieces such that half of the pieces are stored with f and half are stored away from f, allowing them to be subtracted away. In fact, we need to break up every flow in such a way because any flow has a potential to cause error to other flows due to hash collision. Below we describe how $rSkt2$ will record the elements of an arbitrary flow f differently from $rSkt$.

Recall from Section II-B that each estimator in the hash table C (or $\bar{C}$) is an array of m units, which may be bits, FM registers or HLL registers. Flow f is hashed to a pair of estimators, $C[h(f)]$ and $\bar{C}[h(f)]$. Different from rSkt, our new idea will not use either of them to record f in its entirety. Instead, we construct a logical primary estimator L_f from the units of $C[h(f)]$ and $\bar{C}[h(f)]$ to record f. L_f is also an array of m units. Its ith unit is taken from the ith unit of either $C[h(f)]$ or $\bar{C}[h(f)]$, with equal probability. Let $g'(f,i)$ be a pseudo-random function taking two input parameters i and f and returning a bit, 0 or 1, with equal probability, where $0 \le i < m$. We define

$$L_f[i] \equiv \begin{cases} C[h(f)][i], & \text{if } g'(f,i) = 0 \\ \bar{C}[h(f)][i], & \text{if } g'(f,i) = 1 \end{cases} \quad (9)$$

When we receive an element e of flow f, it is recorded as usual in $L_f[h(e)]$, which is $C[h(f)][h(e)]$ if $g'(f,h(e)) = 0$ or $\bar{C}[h(f)][h(e)]$ if $g'(f,h(e)) = 1$. The actual recording operation is explained in Section II-B.

The logical complement estimator of flow f, denoted as $\bar{L}_f$, is constructed from the units that L_f does not use.

$$\bar{L}_f[i] \equiv \begin{cases} \bar{C}[h(f)][i], & \text{if } g'(f,i) = 0 \\ C[h(f)][i], & \text{if } g'(f,i) = 1 \end{cases} \quad (10)$$

Consider an arbitrary flow $f' \in F_f - \{f\}$, where $h(f') = h(f)$ by definition. The flow is recorded in its own logical primary estimator $L_{f'}$ that is constructed similarly from the units of $C[h(f)]$ and $\bar{C}[h(f)]$. Each unit from $C[h(f)]$ or $\bar{C}[h(f)]$ has 50% chance to be in L_f and also independently 50% chance to be in L'_f. Hence, when an element e' of f' is recorded in a unit of L'_f, it has 50% chance to be in L_f as well because that unit has 50% chance to be in L_f. By the same token, element e' has 50% chance to be in $\bar{L}_f$. Hence, we are successful in splitting f' to two halves. One half is stored in L_f, and the other half in $\bar{L}_f$, allowing us to subtract them away. We estimate the spread of flow f based on its logical primary estimator and the logical complement estimator as follows:

$$\hat{s}_f = V(L_f) - V(\bar{L}_f), \quad (11)$$

which not only solves the accuracy problem raised at the beginning of this subsection, but does so with a low query overhead of computing $V(.)$ only twice. Moreover, each element is recorded for d times in rSkt, and it is recorded just once in rSkt2. This smaller processing overhead allows rSkt2 to handle an incoming stream of packets at higher throughput. The recording and querying operations of rSkt2 are formally presented in Algorithms 4 and 5, respectively.

Our experimental results show that rSkt2 significantly improves the estimation accuracy compared to rSkt/rSkt1.

Algorithm 4 Recoding a Packet in rSkt2

```
Input: packet ⟨f, e⟩
Action: record e of f in hash tables C and C̄
if g'(f, h(e)) = 0 then
    record ⟨f, e⟩ to C[h(f)][h(e)]
else
    record ⟨f, e⟩ to C̄[h(f)][h(e)]
end if
```

Algorithm 5 Querying on a Flow in rSkt2

```
Input: flow label f
Output: spread estimate
for i= 0 to m do
    if g'(f, i) = 0 then
        L_f[i] = C[h(f)][i]
        L̄_f[i] = C̄[h(f)][i]
    else
        L_f[i] = C̄[h(f)][i]
        L̄_f[i] = C[h(f)][i]
    end if
end for
return V(L_f) − V(L̄_f)
```

One may expect that rSkt2 can be further optimized by the design idea of rSkt→rSkt1 in Figure 5, in order to support fast query. In other words, *can we maintain an additional integer/float for each estimator, so that $V(L_f)$/$V(\bar{L}_f)$ in* (11) *is obtained by accessing only the respective integer/float instead of all the m units in the estimator?* Unfortunately, we claim it is infeasible and the reason is that the estimator for each flow in rSkt2 is logical and distinct, resulting in the number of logical estimators equal to the number of flows. Since we cannot assign each flow an integer/float, we cannot maintain an additional integer/float for each logical estimator.

E. Network-Wide Measurement

The proposed randomized error reduction sketches can be deployed on multiple locations to jointly measure multiple packets concurrently [9], [26]. For example, it may be deployed on multiple routers to support network-wide flow spread monitoring. Suppose there are k measurement points in a network, each running an instance of rSkt2 with the same parameter setting, e.g., d, w and hash functions. The recorded hash tables, C_j and $\bar{C}_j$, $0 \le j < k$, are sent to a central controller for merging together into two tables, C_* and $\bar{C}_*$. The merge operation is dependent on the estimator type.

- bitmap or FM: Bitmap/FM estimators from k routers, $C_j[i]$, $0 \le i < k$, are merged to $C_*[i]$ by bitwise OR. $\bar{C}_j[i]$, $0 \le i < k$, are merged to $\bar{C}_*[i]$ by bitwise OR.
- HLL: HLL estimators from k routers, $C_j[i]$, $0 \le i < k$, are merged to $C_*[i]$ by taking the maximum unit values, i.e., $C_*[i][z] = \max\{C_j[i][z], 0 \le j < k\}$, $0 \le z < m$. Likewise, $\bar{C}_*[i][z] = \max\{\bar{C}_j[i][z], 0 \le j < k\}$, $0 \le z < m$.

After merging, spread estimation is performed on C_* and $\bar{C}_*$ as described earlier in the section.

Our sketches usually require the amount of memory that is much less than what programmable switch can offer. Specifically, our sketches consumes a memory allocation of 10^1Mb (default setting for memory in the experiments is 2Mb). The operations that randomized sketches need for recording and query are primarily hash, addition, max and bitwise AND, which are supported by programmable switches.

IV. Analysis

Let s_f be the actual spread of flow f, $\hat{s}_f$ be the spread estimate of flow f, S be the total number of distinct packets $\langle f, e\rangle$ in the packet stream, and F_f be the set of flows f' such that $h(f') = h(f)$ in case of rSkt2 or that $\exists i \in [0, d)$, $h_i(f') = h_i(f)$, in case of rSkt/rSkt1. The packets in flow $f' \in F_f - \{f\}$ are called *error packets* with respect to f. The value of m for bitmap, FM and HLL is set as recommended in Table I, where $m \geq 16$ usually holds.

Theorem 1: For any given flow f, supposing $m \geq 16$, the expectation of $\hat{s}_f$ produced by rSkt/rSkt1/rSkt2 satisfies

$$|E(\hat{s}_f) - s_f| \leq \begin{cases} o(s_f) + o(\frac{S-s_f}{2w}), \\ \quad \text{if using HLL/FM estimators;} \\ \frac{s_f}{2m} + \frac{1}{2}(e^{\frac{s_f}{m}} - 1) + o(\frac{S-s_f}{2w}), \\ \quad \text{if using bitmap estimators.} \end{cases}$$

The proof can be found in the supplementary materials.

Note that w is a large value: when using HLL estimators with 128 units and allocated 10Mb memory, w is about 8k. The bound for expectation of the spread estimate produced by rSkt/rSkt2 can be small when f's spread is large. For instance, $\frac{S-s_f}{2w}$ represents the average error in each estimator. If it is smaller than s_f, the bound will become $o(s_f) \ll s_f$. Consider a special case where each flow is allocated an estimator and $2w$ approaches the number of flows. If using HLL estimators, the bound is $o(s_f) + o(\frac{S-s_f}{2w})$. $\frac{S-s_f}{2w}$ is equal to the average flow spread among all flows. Under this circumstance, $|E(\hat{s}_f) - s_f| \leq o(s_f)$ if f' spread is above the average flow spread, which is much smaller compared to its actual flow spread.

Theorem 2: For any given flow f, supposing $m \geq 16$, the variance of the spread estimate $\hat{s}_f$ from rSkt/rSkt1 can be derived as

$$Var(\hat{s}_f) = \begin{cases} \frac{1.04^2}{m}(s_f^2 + \frac{(S-s_f)s_f}{w} + T_f) + Q_f + o(s_f^2 + Q_f), \\ \quad \text{if using HLL estimators;} \\ \frac{0.78^2}{m}(s_f^2 + \frac{(S-s_f)s_f}{w} + T_f) + Q_f + o(s_f^2 + Q_f), \\ \quad \text{if using FM estimators;} \\ (T_f - (\frac{S-s_f}{2w})^2)(\lambda_1+\lambda_5)^2 + \lambda_4 + \lambda_8 + (\lambda_1+\lambda_5)o(\frac{Q_f}{w}) \\ \quad \text{if using bitmap estimators.} \end{cases} \quad (12)$$

where $\beta = S - s_f$, $Q_f = \sum_{f'\neq f}\sum_{f''\neq f', f''\neq f} s_{f'}s_{f''}(1-(1-\frac{1}{w})^d)^2 + \sum_{f'\neq f} s_{f'}^2(1-(1-\frac{1}{w})^d)$, $T_f = \sum_{f'\neq f}\sum_{f''\neq f', f''\neq f}\frac{s_{f'}s_{f''}}{4}(1-(1-\frac{1}{w})^d)^2 + \sum_{f'\neq f}\frac{s_{f'}^2}{2}(1-(1-\frac{1}{w})^d)$, $\lambda_1 = (\frac{e^{\frac{s_f+\frac{\beta}{2w}}{m}}-1}{2m}+1)$, $\lambda_4 = m(e^{\frac{s_f+\frac{\beta}{2w}}{m}} - \frac{s_f+\frac{\beta}{2w}}{m} - 1)$, $\lambda_5 = (\frac{e^{\frac{\beta}{2mw}}-1}{2m}+1)$, and $\lambda_8 = m(e^{\frac{\beta}{2mw}} - \frac{\beta}{2mw} - 1)$.

Theorem 3: For any given flow f, supposing $m \geq 16$, the variance of the spread estimate $\hat{s}_f$ from rSkt2 can be derived as

$$Var(\hat{s}_f) = \begin{cases} \frac{1.04^2}{m}(s_f^2 + \frac{\beta(s_f+\frac{1}{2})}{w} + \frac{R_f}{2}) + \frac{\beta}{w} + o(s_f^2 + R_f), \\ \quad \text{if using HLL estimators;} \\ \frac{0.78^2}{m}(s_f^2 + \frac{\beta(s_f+\frac{1}{2})}{w} + \frac{R_f}{2}) + \frac{\beta}{w} + o(s_f^2 + R_f), \\ \quad \text{if using FM estimators;} \\ \frac{S-s_f}{4w}(\lambda_1+\lambda_5)^2 + \lambda_4 + \lambda_8 + (\lambda_1+\lambda_5)o(\frac{S}{w}), \\ \quad \text{if using bitmap estimators.} \end{cases} \quad (13)$$

where $R_f = \sum_{f'\neq f}\sum_{f''\neq f', f''\neq f}\frac{s_{f'}s_{f''}}{w^2} + \sum_{f'\neq f}\frac{s_{f'}^2}{w}$ and $\beta, \lambda_1, \lambda_4, \lambda_5, \lambda_8$ are the same as those in Theorem 2.

The proofs of Theorems 3 and 2 are provided in the supplementary materials.

Interpretation of Theorems 2 and 3. After the rigorous derivation of the variances of the spread estimate produced by rSkt/rSkt1/rSkt2. We interpret Theorems 2 and 3 with some approximation. We first do approximation on R_f.

$$\begin{aligned} R_f &= \sum_{f'\neq f}\sum_{f''\neq f', f''\neq f}\frac{s_{f'}s_{f''}}{w^2} + \sum_{f'\neq f}\frac{s_{f'}^2}{w} \\ &= \sum_{f'\neq f}\sum_{f''\neq f', f''\neq f}\frac{s_{f'}s_{f''}}{w^2} + \sum_{f'\neq f}\frac{s_{f'}^2}{w^2} - \sum_{f'\neq f}\frac{s_{f'}^2}{w^2} + \sum_{f'\neq f}\frac{s_{f'}^2}{w} \\ &\leq \frac{(\sum_{f'\neq f} s_{f'})^2}{w^2} + n(\frac{S-s_f}{n})^2\frac{1}{w}) \leq 2(\frac{S-s_f}{w})^2 \end{aligned} \quad (14)$$

where n is the number of flows. The last inequality holds as $n \geq w$ usually holds in practical settings. Similarly, we can do approximation on Q_f and T_f.

$$Q_f \approx ((S-s_f)(1-(1-\frac{1}{w})^d))^2 \approx 2(\frac{d(S-s_f)}{w})^2 \quad (15)$$

$$T_f \approx (\frac{d(S-s_f)}{w})^2 \quad (16)$$

Consider approximation on λ_1. $s_f + \frac{\beta}{2w}$ represents the expected spread stored in f' primary estimator, which is $O(m)$ as the estimation upper bound of bitmap (also called linear counting) is linear to the bitmap length m. Therefore, we have

$$\lambda_1 = (\frac{e^{\frac{s_f+\frac{\beta}{2w}}{m}}-1}{2m}+1) \approx (\frac{1+\frac{s_f+\frac{\beta}{2w}}{m}-1}{2m}+1) \approx 1$$

Doing the similar approximation on λ_5, λ_4, and λ_8, we have

$$\begin{aligned} \lambda_5 &\approx 1 \\ \lambda_4 &= m(e^{\frac{s_f+\frac{\beta}{2w}}{m}} - \frac{s_f+\frac{\beta}{2w}}{m} - 1) \approx \frac{(s_f+\frac{\beta}{2w})^2}{2m} \\ \lambda_8 &= m(e^{\frac{\beta}{2mw}} - \frac{\beta}{2mw} - 1) \approx \frac{1}{2}m(\frac{\beta}{2mw})^2 \approx 0 \end{aligned}$$

From the above approximations, we give a concise version of the variance of estimate produced by rSkt/rSkt1/rSkt2.

• *For any given flow f, the rigorous variance of the spread estimate $\hat{s}_f$ from rSkt/rSkt1 in* (12) *can be approximately*

bounded as

$$Var(\hat{s}_f) \leq \begin{cases} \frac{1.04^2}{m}(s_f + \frac{d(S-s_f)}{w})^2 + 2(\frac{d(S-s_f)}{w})^2, \\ \text{if using HLL estimators;} \\ \frac{0.78^2}{m}(s_f + \frac{d(S-s_f)}{w})^2 + 2(\frac{d(S-s_f)}{w})^2, \\ \text{if using FM estimators;} \\ \frac{1}{2m}(s_f + \frac{S-s_f}{2w})^2 + 4(4d^2-1)(\frac{(S-s_f)}{2w})^2, \\ \text{if using bitmap estimators.} \end{cases} \quad (17)$$

• *For rSkt2, variance in* (13) *can be approximately bounded as*

$$Var(\hat{s}_f) \leq \frac{c^2}{m}(s_f + \frac{(S-s_f)}{w})^2 + \frac{(S-s_f)}{w} \quad (18)$$

where c is 1.04, 0.78, and $1/\sqrt{2}$ *if using HLL, FM, and bitmap estimators, respectively.*

Comparing (18) with (17), we can find the variance of spread estimate produced by rSkt2 is smaller than that of rSkt/rSkt1. Consider (18) for rSkt2. $\frac{S-s_f}{w}$ can be interpreted as the average error in the each pair of candidate estimators. When the spread of flow f is far larger than the average error, the variance is bounded by $\frac{c^2}{m}(s_f)^2 + \frac{(S-s_f)}{w}$. When the spread of flow f is far smaller than the average error, the variance is bounded by $\frac{(S-s_f)}{w}$.

V. Performance Evaluation

We evaluate the performance of the proposed rSkt, rSkt1, and rSkt2 on both hardware and software platforms through experiments based on real-world data traces. We also compare them with the state of the art under various performance metrics. In addition, we perform an application case study on super spread detection in comparison with the prior art.

A. Implementation

We have implemented the following sketches: (1) the proposed sketches, rSkt, rSkt1 and rSkt2, (2) the state-of-the-art prior work that performs per-flow spread estimation, bSkt [9] and cSkt-CM [9], [26], and (3) the state-of-the-art prior work that performs super spreader detection, SS [35]. SS uses multi-resolution bitmaps [3]. The other sketches can work with bitmaps, FM estimators, and HLL estimators, which are explained in Section II-B. With different estimators, they are denoted respectively as rSkt(bitmap), rSkt(FM), and rSkt(HLL); rSkt1(bitmap), rSkt1(FM), and rSkt1(HLL); rSkt2(bitmap), rSkt2(FM), and rSkt2(HLL); bSkt(bit-map), bSkt(FM), and bSkt(HLL); cSkt-CM(bitmap), cSkt-CM(FM), and cSkt-CM(HLL). We also compare the proposed sketches with AROMA [41] and CountMin based solution with a error removal technique [47], denoted as CM-ER. AROMA is the state of the art on spread estimation with a flexible memory-accuracy tradeoff— with more memory allocated, more flows will be measured and the accuracy will be improved. CM-ER tries to remove the estimation bias caused by the hash collision among flows. It maintain an additional spread estimator (i.e., HLL) to estimate the total spread and calculate the expected error by [47] that should be subtracted from the estimate. Our implementation is done on three platforms. *CPU Implementation*: This is software implementation. The experiments are performed on a computer with Intel Core Xeon W-2135 3.7GHz and 32 GB memory. *GPU Implementation*: GPU has become cheaper and widely available. We find that it serves well as a low-cost accelerator for software implementation. With CUDA toolkit, all sketches are programmed to support parallel execution on a computer equipped with GeForce GTX 1070, 8GB GDDR5 memory and 1920 CUDA cores, each at a rate of 1506-1683 MHz clock rate. *FPGA Implementation*: This is hardware implementation. All sketches are implemented on XILINX NEXYS 4DDR/A7 -100T FPGA platform, with 128MB DDR2 DRAM, 4860Kb Block RAM, and 100MHz clock rate.

B. Experimental Setting

The packet streams used in our evaluation are real Internet traffic traces downloaded from CAIDA [48]. We use 10 traces, each of tens of millions of packets. Each experiment is performed over these 10 packet streams independently, and we present the average results. Flow label f is defined as destination address carried in each packet's header. Each trace contains around 110k flows and around 400k distinct packets. Element e is source address also from packet header. All packets toward the same destination form a flow. Flow spread is the number of distinct sources that communicate with a destination. Anomaly in flow spread may signal flash crowd in service requests or denial-of-service attack against a destination service (which could be judged in conjunction with flow size); both cases will require immediate attention from service admin.

For rSkt(bitmap), rSkt2(bitmap), bSkt(bitmap) and cSkt-CM(bit-map), we set the bitmap size to be 5000 bits, which produces a spread estimation range that covers all flows in the traces. The bitmap size of SS is chosen according to the original paper [35]. For rSkt(FM), rSkt2(FM), bSkt(FM) and cSkt-CM(FM), each register is 32 bits. For rSkt(HLL), rSkt2(HLL), bSkt(HLL) and cSkt-CM(HLL), each register is 5 bits. The number of registers in each estimator is 128. For rSkt, bSkt and cSkt-CM, $d = 4$. Namely, each packet is recorded in four estimators and each query requires estimation from four estimators; see Section III-B. The above parameter setting is in line with those in [9].

The sketches are evaluated and compared under the following four performance metrics. *Estimation Accuracy*. We use absolute error to measure estimation accuracy. Let $\hat{s}_f$ be the estimated spread of flow f, and s_f be the actual spread of flow f. The absolute error is calculated as $|\hat{s}_f - s_f|$, and the average absolute error is defined as $\sum_f |\hat{s}_f - s_f|/N$, where N is the number of flows in the packet stream. *Recording Throughput*. We measure the rate at which the packet $\langle f, e\rangle$ are recorded by each sketch on any given software/hardware platform. The unit is million packets per second, abbreviated as Mpps. *Online Query Throughput*. We measure the rate at which the queries can be performed on f after each packet $\langle f, e\rangle$ from a stream is recorded. For each query, we produce an estimate of f's spread up to the time when the query is performed. *Query Overhead*. It is to measure the processing time to answer one query on any flow's spread. The unit is ms.

TABLE IV
PROCESSING TIME (MS) PER QUERY BY VSKT, CSE, VHLL, AROMA, AROMA+, BSKT, CSKT-CM, CM-ER, RSKT, RSKT1, AND RSKT2 USING DIFFERENT SINGLE-FLOW SPREAD ESTIMATORS, INCLUDING BITMAP, FM AND HLL. NOTE THAT AROMA DOES NOT USE BITMAP, FM, AND HLL AS PLUG-INS. AROMA+ IS THE MODIFIED VERSION OF AROMA FOR ONLINE SPREAD ESTIMATION. IN COMPARISON, VSKT, CSE AND VHLL AND AROMA HAVE MUCH HIGHER QUERY OVERHEAD THAN OTHERS

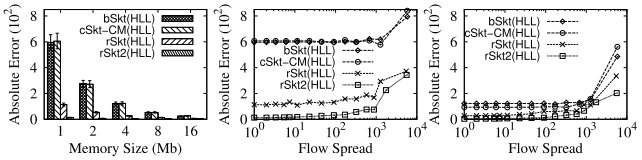
Sketches \ Plug-ins	HLL	FM	bitmap
vSkt [9]	9.260	0.376	1.644
CSE [6]	-	-	1.745
vHLL [28]	7.299	-	-
AROMA [41], [42]	0.100		
AROMA+	0.001		
bSkt [9]	0.005	0.002	0.060
cSkt-CM [9], [26], [27]	0.005	0.002	0.060
CM-ER [47]	0.006	0.003	0.064
rSkt	0.008	0.002	0.120
rSkt1	0.001	0.001	0.001
rSkt2	0.002	0.001	0.030

C. Query Overhead

We compare two groups of spread estimation solutions in terms of the query overhead. The first group includes bSkt, cSkt-CM, CM-ER, rSkt, rSkt1, and rSkt2. The second group includes vSkt, CSE, vHLL and AROMA. We set the memory as 1Mb under which AROMA have similar accuracy performance to our most accurate sketches, i.e., rSkt2(HLL) and rSkt2(bitmap), which will be explained later. Note that AROMA can be modified reduce its query overhead. Specifically, we can maintain a hash table to store the number of samples for each flow and maintain an additional HLL to estimate the total spread, which is used to calculate the sampling rate. In our experiment, we find the memory of hash table and an HLL takes up around 10% of the total memory. The modified version is denoted as *AROMA+*. The results on the processing time per query are given in Table IV. Among the second group of solutions, AROMA has the lowest processing time, which however is far higher than what the first group can achieve. Specifically, AROMA needs a processing time of 0.10 ms to query the spread of one flow, 50 times larger than what bSkt, cSkt-CM, CM-ER, rSkt, rSkt1 and rSkt2 need ($\leq$0.002 ms when the proper single-flow spread estimators are used as plug-ins). The reason is that the second group of solutions all need to scan the whole data structure to answer the online queries on the spread of one flow, while the first group of solutions only need to access $O(d)$ single-flow spread estimators or even 2 (for rSkt2). Since this paper focuses on online spread estimation, in the rest of the evaluation, we will focus on comparing the first group of solutions and AROMA (and AROMA+) as it can be easily modified to support fast online queries.

D. Estimation Accuracy

We first compare randomized sketches with cSkt, CM and bSkt comprehensively using various types of spread estimators, i.e., HLL, FM and bitmap. Then we compare randomized sketches with the existing error removal method, i.e., CM-ER. Finally, we select our most accurate ones and compare them with the state-of-the-art on spread estimation, i.e., AROMA.



(a) Various memory (b) 1Mb Memory (c) 4Mb Memory

Fig. 6. Accuracy comparison of rSkt(HLL), rSkt2(HLL), bSkt(HLL) and cSkt-CM(HLL) under CAIDA dataset. (a) Average absolute error of all flows w.r.t memory size, (b)-(c) error distribution under a given memory size. Compared to the prior art bSkt(HLL) and cSkt-CM(HLL), the proposed rSkt(HLL) and rSkt2(HLL) reduce absolute error by 73.6%-81.1% and 93.9%-97.8%, respectively, in plot (a).

1) Compare Randomized Sketches With cSkt, CM and bSkt: Our first set of experiments compare the proposed sketches with the state of the art in terms of estimation accuracy. Note that accuracy is the same across different implementation platforms, which only affect throughput. *We want to stress that the estimation accuracy of rSkt and rSkt1 are the same under the same values of w and d.* As we have explained in Section III-C, compared to rSkt, rSkt1 increases the memory consumption by $<5\%$ under the same w, which can be neglected. Therefore, we do not plot the estimation accuracy results of rSkt1 in the figure and only compare the throughput results in the following subsections. We begin by comparing rSkt(HLL), rSkt2(HLL), bSkt(HLL) and cSkt-CM(HLL). Figure 6 (a) shows the average absolute error among all flows under 1Mb-16Mb memory allocations to each sketch. In contrast, if we ideally assign each flow a single-flow spread estimator, it needs 70Mb/450Mb/550Mb memory using HLL, FM, and bitmap estimators, respectively. bSkt(HLL) and cSkt-CM(HLL) performs similarly. Compared to the better one of them, rSkt(HLL) reduces average error by more than 73.6%, and rSkt2(HLL) reduces average error by more than 93.9%. Figures 6(b)-6(c) show the detailed error distribution at a given memory allocation, 1Mb and 4Mb, respectively. The flows are placed in bins based on their true spreads (which can be found directly from the traffic traces). The spread bins are $[2^i, 2^{i+1}]$ for $i \geq 0$. We average the absolute error of flows in each bin and plot a point in the figure.

In Figure 6(a), when memory allocation increases, the average absolute error of rSkt(HLL), rSkt2(HLL), bSkt(HLL) or cSkt-CM(H-LL) decreases, which is expected because the probability of hash collision decreases. The figure shows that rSkt and rSkt2 are much more accurate than bSkt(HLL) and cSkt-CM(HLL), especially under tight memory. For example, when 1Mb memory is used, rSkt(H-LL) and rSkt2(HLL) reduce the average absolute error by 81.1% and 97.8%, respectively, compared to bSkt(HLL). Figure 6(a) also shows the error bars of average absolute error of rSkt(HLL), rSkt2(HLL), bSkt(HLL) and cSkt-CM(HLL) under 10 traces. As we can see, the advantages of rSkt(HLL), rSkt2(HLL) over bSkt(HLL) and cSkt-CM(HLL) in terms of estimation accuracy hold under different traces.

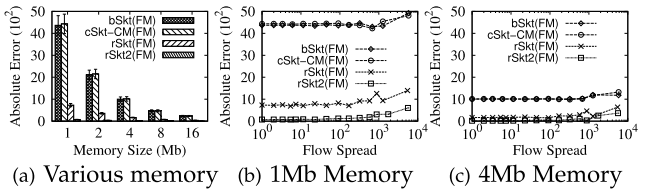


Fig. 7. Accuracy comparison of rSkt(FM), rSkt2(FM), bSkt(FM) and cSkt-CM(FM) under CAIDA dataset. (a) Average absolute error of all flows w.r.t memory size, (b)-(c) error distribution under a given memory size. Compared to the prior art bSkt(FM) and cSkt-CM(FM), the proposed rSkt(FM) and rSkt2(FM) reduce absolute error by 83.5%-84.5% and 97.9%-98.4%, respectively, in plot (a).

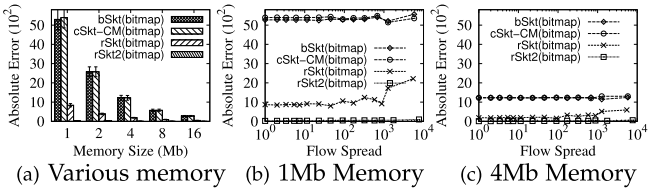


Fig. 8. Accuracy comparison of rSkt(bitmap), rSkt2(bitmap), bSkt(bitmap) and cSkt-CM(bitmap) under CAIDA dataset. (a) Average absolute error of all flows w.r.t memory size, (b)-(c) error distribution under a given memory size. Compared to the prior art bSkt(bitmap) and cSkt-CM(bitmap), the proposed rSkt(bitmap) and rSkt2(bitmap) reduce absolute error by 83.0%-84.5% and 98.7%-99.5%, respectively, in plot (a).

Figures 6(b)-6(c) show that absolute error is larger for flows of larger spreads. The proposed rSkt(HLL) and rSkt2(HLL) have much smaller error distributions than bSkt(HLL) and cSkt-CM(HLL), thanks to their randomized error reduction design. rSkt2(HLL) is more accurate than rSkt(HLL) due to its logical estimator design that splits noise flows into pieces. Its improvement over rSkt will be more pronounced when we use FM estimators and bitmaps below and when we consider throughput shortly.

The experimental results that compare rSkt(FM), rSkt2(FM), bSkt-(FM) and cSkt-CM(FM) are shown in Figure 7. The results that compare rSkt(bitmap), rSkt2(bitmap), bSkt(bitmap) and cSkt-CM(bit-map) are shown in Figure 8. Similar conclusion can be drawn as those from Figure 6. For example, from Figure 7(a), using bSkt(FM) as a baseline, rSkt(FM) reduces average error by more than 83.5%, and rSkt2(FM) reduces average error by more than 97.9%. From Figure 8 (a), using bSkt(bitmap) as a baseline, rSkt(bitmap) reduces average error by more than 83.0%, and rSkt2(bitmap) reduces average error by more than 98.7%. From error distributions in Figure 7 (b)-(c) and Figure 8(b)-(c), rSkt2 performs consistently better than rSkt, which is in turn much better than bSkt and cSkt-CM.

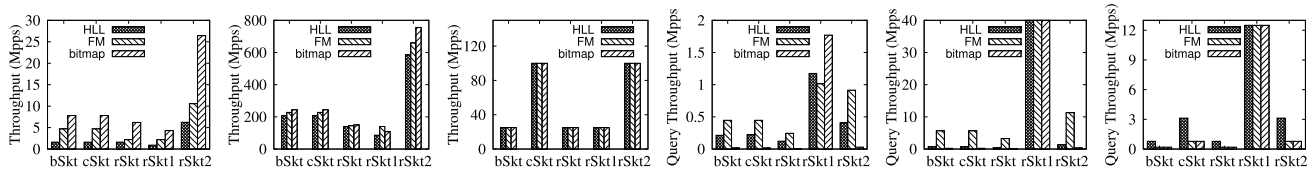
2) Compare Randomized Sketches With AROMA and CM-ER: From the above accuracy comparison, we find that rSkt2(HLL) and rSkt2(bitmap) are the best randomized sketches in terms of accuracy performance. Therefore, we compare them with AROMA (and AROMA+), CM-ER(HLL) and CM-ER(bitmap). AROMA is the state of the art on spread estimation with a flexible memory-accuracy tradeoff and AROMA+ is the modified version of AROMA to support online queries, which has been explained in Section V-C. CM-ER is based on CM but applies an error removal technique from [47]. It can be plugged in various types of spread estimators, e.g., HLL and bitmap, and is denoted as CM-ER(HLL) and CM-ER(bitmap), respectively. The parameter settings for HLL and bitmap are consistent for both rSkt2 and CM-ER. We first set memory allocation for each solution to 1Mb and then observe the average absolute error of flows with respect to the actual flow spread. Note that the line for AROMA starts from $x = 1/p$, where p is the sampling rate, meaning that most flows with spread less than $1/p$ will not be sampled. The results in Figure 10 show that rSkt2(HLL) is the most accurate for small flows, whereas AROMA samples out most small flows (see Figure 10(a)) and show that rSkt2(bitmap) is the most accurate for large flows, whereas AROMA outperforms rSkt2(HLL) for large flows (see Figure 10(b)). AROMA+ is slightly less accurate than AROMA as it needs around 10% of the memory to maintain a hash table and an HLL. When we increase the memory allocation to 4Mb, the same conclusion can be drawn from Figure 11. What's more, rSkt2 will always outperforms CM-ER, reducing the error by at least 66% using various types of spread estimators.

E. Recording Throughput

1) Compare Randomized Sketches With cSkt, CM and bSkt: Our second set of experiments compare the proposed sketches with the state of the art in terms of recording throughput (at which rate the incoming packets can be processed on different platforms). The experimental results of recording throughput on the CPU platform are shown in Figure 9(a). The recording throughput of rSkt2 is highest for any type of estimators (i.e., bitmap, FM and HLL) because it records each packet just once, whereas the other three sketches records each packet d times. The throughputs of bSkt and rSkt are similar, while that of rSkt is slightly lower due to computing an additional function g and that of rSkt1 is also slightly lower due to maintaining additional integer/float arrays. As example, the throughput of rSkt2(bitmap) is 3.33 times that of bSkt(bitmap) or cSkt-CM(bitmap), and it is 4.19 times that of rSkt(bitmap) and 6.04 times that of rSkt1(bitmap). For all sketches, the highest throughput is achieved when bitmaps are used. That is because FM and HLL require an additional geometric hash operation; see Section II-B. The throughput is lowest when HLL is used because it incurs more memory accesses.

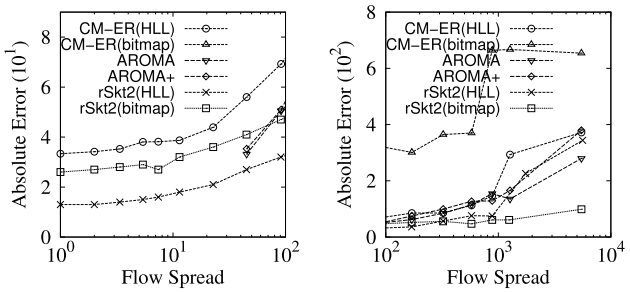
The experimental results of recording throughput on the GPU platform are shown in Figure 9(b). All sketches achieve much higher throughput on GPU than on CPU due to massive parallelism. Still, rSkt2 achieves much higher throughput, around 600 Mpps, about three that of bSkt or cSkt-CM and about four times that of rSkt and rSkt1.

The recording throughput of the sketches on FPGA is shown in Figure 9(c). Note that rSkt1(HLL) needs a float for each HLL estimator. For ease of implementation, we multiply the float by 2^{32}, such that the value becomes integer. The proposed rSkt2 achieves a throughput of 100 Mdps, while bSkt, rSkt and rSkt1 only support a throughput of 25 Mdps. This is because bSkt, rSkt and rSkt1 record each packet four times in the same memory block, which consumes four clock cycles, whereas rSkt2 records each packet once in one clock



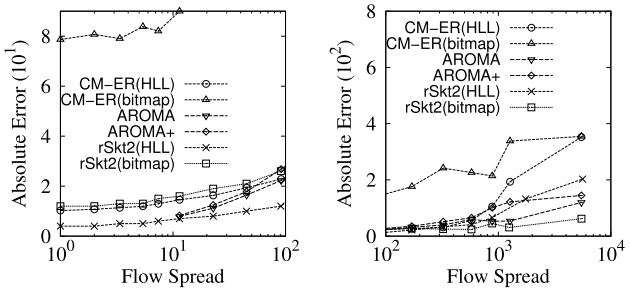
(a) CPU (b) GPU (c) FPGA (d) CPU (e) GPU (f) FPGA

Fig. 9. Plots (a)-(c): Recording throughput on CPU, GPU and FPGA platforms. Plots (d)-(f): Online query throughput on CPU, GPU and FPGA platforms. Mpps stands for Mega packets per second.



(a) Flows with spread <100 (b) Flows with spread ≥100

Fig. 10. Accuracy comparison of CM-ER(HLL), CM-ER(bitmap), AROMA, AROMA+, rSkt2(bitmap) and rSkt2(HLL) with respect to flow spread under 1Mb memory allocation. Plot (a): error of small flows whose spreads are smaller than 100; Plot (b): error of flows whose spreads are equal to or larger than 100. rSkt2(bitmap) achieves the best accuracy among large flows, while rSkt2(HLL) performs best for small flows. Using the same spread estimators, rSkt2 always outperforms CM-ER. Note that AROMA starts from $x \approx 45$ as its sampling rate under 1Mb memory is $\frac{1}{45}$, meaning that most flows with spread less than 40 will not be sampled.



(a) Flows with spread <100 (b) Flows with spread ≥100

Fig. 11. Accuracy comparison of CM-ER(HLL), CM-ER(bitmap), AROMA, AROMA+, rSkt2(bitmap) and rSkt2(HLL) with respect to flow spread under 4Mb memory allocation. Plot (a): error of small flows whose spreads are smaller than 100; Plot (b): error of flows whose spreads are equal to or larger than 100. Same conclusion from Fig. 10 can be drawn. Note that AROMA starts from $x \approx 11$ as its sampling rate under 4Mb memory is $\frac{1}{11}$, meaning that most flows with spread less than 10 will not be sampled.

cycle (with hardware pipelining). Interestingly, cSkt-CM also achieves 100 Mpps because it uses d arrays, which can be placed on different memory blocks, allowing parallel access. Due to pipelining, each sketch achieves the same throughput under different estimator types (bit-map, FM and HLL). One may observe that hardware implementation on FPGA achieves smaller throughput than software implementation on GPU. There are two reasons. First, GPU allows massive parallelism which compensates the software disadvantage. Second, our FPGA is a cheap one. Throughput will be higher if a high-end FPGA is used. We conclude that GPU is a viable alternative to hardware implementation for high throughput.

TABLE V
RECORDING THROUGHPUT (MPPS) OF AROMA, AROMA+ AND CM-ER, RSKT2

	CM-ER(HLL)	CM-ER(FM)	CM-ER(bitmap)	AROMA+
Th.	1.3	4.0	6.2	3.8
	rSkt2(HLL)	rSkt2(FM)	rSkt2(bitmap)	AROMA
Th.	6.2	10.5	26.4	40.4

2) Compare Randomized Sketches With AROMA and CM-ER: We also compare our best sketches, i.e., rSkt2(HLL) and rSkt2(bitmap) with AROMA, AROMA+ (modified version to support online queries) and CM-ER. The results in Table V show that rSkt2 rSkt2(HLL) and rSkt2(bitmap) record packets much faster than CM-ER. The reason is that CM-ER needs to record each packet d times while rSkt2 only needs to record once. Among all, AROMA has the highest recording throughput, but we will show that its query throughput is very limited. Its online version, i.e., AROMA+ records packets slightly slower than rSkt2. The reason is that it needs to maintain the number of samples and an additional HLL for total spread estimation.

F. Query Throughput

Our third set of experiments compare the proposed sketches with the state of the art in terms of query throughput. We want to stress that the computation of spread estimation is nothing similar to that of size estimation [30], [32], [34], [49]. The latter incurs similar overhead as recording, and therefore its query throughput is similar to recording throughput. But spread estimation is much more computation intensive than recording, and spread query throughput is much smaller than recording throughput. Queries cannot be performed on per packet basis, which makes it practically important to design novel sketches that improve on query throughput.

The experimental results of query throughput on the CPU platform are shown in Figure 9(d). As is expected, the query throughput of rSkt1 is the highest for any type of estimators (i.e., bitmap, FM and HLL) because we maintain an additional integer/float for each estimator, without accessing m units compared to other sketches. Moreover, its query throughput is 100%, 50%, and 40% of its recording throughput when using HLL, FM and bitmap estimators, respectively. That means, rSkt1 can almost support per-two-packet online query or even per-packet online query if using HLL estimators. Aside from rSkt1, the query throughput of rSkt2 is highest because it computes from two estimators per query, whereas bSkt and cSkt-CM each compute from d estimators per query, while rSkt computes $2d$ estimators. Because $d = 4$ in our

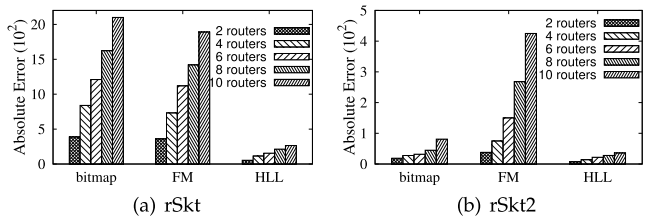


Fig. 12. Network-wide measurement estimation accuracy of rSkt and rSkt2 using different plug-in estimators.

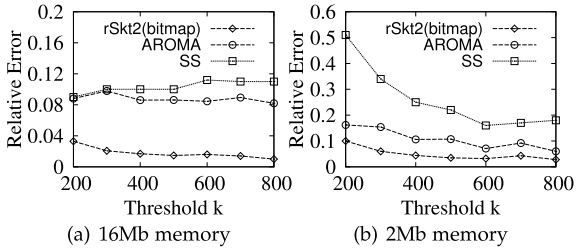


Fig. 13. rSkt2(bitmap) incurs smaller relative error than SS and AROMA, particularly when memory allocation is small.

experiments, the throughput of rSkt2 is expected to be about twice that of bSkt (or cSkt-CM) and about four times that of rSkt, matching well with the experimental results.

The experimental results of query throughput on the GPU platform are shown in Figure 9(e). Again, GPU is a great accelerator thanks to its numerous cores that process in parallel. The query throughput on GPU is more than an order of magnitude higher than that on CPU across different sketches and different types of estimators. Yet, relative performance between sketches remain similar. The query throughput of rSkt1 is the highest (124 Mpps, 120 Mpps, and 132 Mpps when HLL, FM, and bitmap estimators are used respectively), ten times more than those of other sketches. The query throughput of rSkt2 is the second highest, about twice that of bSkt (or cSkt-CM) and about four times that of rSkt.

The complexity of query computation is far greater than that of recording, particularly for FM estimators and HLL estimators, which prevent us from implementing query solely on the FPGA board that we have. Instead, we implement a module that, upon query, will output the estimators of the flow for bSkt, cSkt-CM, rSkt and rSkt2 and the corresponding integers for rSkt1, from which one can compute spread estimation off-board by software (which may be GPU-accelerated) or by ASIC hardware. The throughput of this FPGA module is shown in Figure 9(f). Due to our maintaining of additional integers for each estimator, rSkt1 can achieve the query throughput of 12.5 Mpps, at least fours time larger that those of other sketches. Aside from rSkt1, both cSkt-CM and rSkt2 achieve higher throughput, thanks to pipelining, as the design of cSkt-CM allows parallel access to its d estimators per flow on FPGA, while rSkt2 also allows parallel access to its 2 estimators per flow. Both bSkt and rSkt have lower throughput because their designs do not allow fully parallelized access to multiple estimators per flow on FPGA.

G. Case Study 1: Network-Wide Measurement

In this subsection, we conduct experiments to evaluate the performance of randomized sketches for network-wide measurement, in terms of the estimation accuracy. *We want to stress again that the estimation accuracy of rSkt1 is the same as that of rSkt under the same value of w and d. Therefore, we do not repeat the results and only present the results of rSkt and rSkt2.* The number of measurement locations, e.g., routers, is up to 10. We combine 10 1-minute CAIDA data trace together as a whole packet stream in the network. For each packet in the whole stream, we assign it to a randomly selected router. The performance metric is average absolute error, which has been defined before. The memory allocation for each router is 4Mb. We vary the number of routers from 2 to 10 to observe the estimation accuracy of rSkt and rSkt2 under different numbers of measurement locations. The results are presented in Figure 12, which demonstrates that with the number of routers increasing (i.e, the total number of packets received increasing), the average absolute error of each sketch increases accordingly. By comparison, rSkt2 outperforms rSkt under the same parameter setting, regardless of the plug-ins used. Among all the plug-ins, rSkt(HLL) and rSkt2(HLL) outperform rSkt(FM) and rSkt(bitmap), and rSkt2(FM) and rSkt2(bitmap), respectively. The best sketch, i.e., rSkt2(HLL) can maintain very low average absolute error, i.e, less than 50, even for network-wide measurement with 10 routers.

H. Case Study 2: Super Spreader Detection

We use a case study to investigate how the proposed sketches perform in detecting super spreaders, which are defined as the flows whose spreads exceed a threshold k that the user chooses based on application need. We have shown that the proposed rSkt2 outperforms the state of the art on per-flow spread estimation. This case study is different. It is to identify super spreaders only and estimate their spreads. The focus of this subsection is to evaluate the detection accuracy. In this experiment, we compare rSkt2(bitmap) with the state-of-the-art sketch for this purpose, SpreadSketch (SS) [35] that uses multi-resolution bitmaps and AROMA [41] that is based on sampling. SS is a modified version of cSkt-CM, with each estimator expanded for storing a flow label. With online queries, if the estimated spread of a flow after element record exceeds k, we keep the flow label in a hash map. For query, AROMA accesses the whole register array after recording all packets to collect the labels of super spreaders and estimate their spreads. The parameter setting can be found in Section V-B. We evaluate the performance with three metrics. *Average relative error*, which is defined as $\sum_{f\in\Gamma_s}\frac{|\hat{s}_f-s_f|}{s_f\cdot|\Gamma_s|}$, where Γ_s is the set of super spreaders detected. *Number of false positives*, i.e., the number of detected "super spreaders" whose true spreads are below k. *Number of false negatives*, i.e., the number of real super spreaders that are not detected.

We perform two experiments with different memory allocations, 16Mb and 2Mb, respectively. Figure 13(a) shows the results under 16Mb. The relative error of SS is in the range of [9.0%,11.2%] and that of AROMA is in the range of [8.1%,9.7%] where the threshold ranges from 200 to 800, whereas rSkt2(bitmap) performs better with relative error between 1.0% and 3.3%. Figure 13(b) shows the results under 2Mb. The relative error of SS is in the range of [17.3%,51.0%]

TABLE VI

NUMBER OF TRUE SUPER SPREADERS UNDER DIFFERENT SUPER SPREADER THRESHOLD

Threshold	200	300	400	500	600	700	800
Number	146	88	55	49	36	34	32

TABLE VII

NUMBER OF FALSE POSITIVES

Threshold	200	300	400	500	600	700	800
rSkt2(bitmap)	43	15	9	3	4	2	0
SS	1122	113	67	27	19	15	6
AROMA	32	22	14	2	5	2	0

TABLE VIII

NUMBER OF FALSE NEGATIVES

Threshold	200	300	400	500	600	700	800
rSkt2(bitmap)	6	2	1	0	0	0	0
SS	12	6	2	1	0	0	0
AROMA	9	11	7	4	1	1	3

and that of AROMA is in the range of [6.2%, 16.2%], whereas rSkt2(bitmap) performs better with relative error between 2.8% and 10.0%. We find that rSkt2(bitmap) works well under tight memory when the performance of SS and AROMA deteriorates. This is also true in terms of false positives and false negatives. Table VI shows the true number of super spreaders in the packet traces that we use in this experiment. Under 2Mb, Table VII shows that SS reports much more false positives than rSkt2(bitmap) and AROMA reports similar number of false positives to rSkt2(bitmap). In practice, more false positives may lead to additional false alarms and take extra time from system admin to investigate. Table VIII shows that SS and AROMA also produce more false negatives than rSkt2(bitmap). In practice, more false negatives may allow some true offenders to escape timely detection.

VI. CONCLUSION

In this paper, we have proposed three randomized error-reduction sketches for online measurement of flow spread. They provide an implementation framework with bitmaps, FM estimators or HLL estimators as plug-ins to meet different performance-overhead requirements. The new sketch designs split error (introduced by other flows due to estimator sharing) into two halves, one stored with the flow of interest in a primary estimator and the other half stored separately in a complement estimator. By subtracting the complement from the primary estimator, we are able to statistically remove the error and achieve an accuracy one order of magnitude better than the prior art. Through theoretical analysis and experimental studies, we show that our randomized sketches work well on both software platform and hardware platform, producing accurate spread estimates in tight memory at low processing overhead for online queries.

REFERENCES

[1] H. Wang, C. Ma, O. O. Odegbile, S. Chen, and J.-K. Peir, "Randomized error removal for online spread estimation in data streaming," *Proc. VLDB Endowment*, vol. 14, no. 6, pp. 1040–1052, Feb. 2021.

[2] Z. Durumeric, M. Bailey, and J. A. Halderman, "An internet-wide view of internet-wide scanning," in *Proc. 23rd USENIX Secur. Symp.*, 2014, pp. 65–78.

[3] C. Estan, G. Varghese, and M. Fisk, "Bitmap algorithms for counting active flows on high-speed links," *IEEE/ACM Trans. Netw.*, vol. 14, no. 5, pp. 925–937, Oct. 2006.

[4] P. Lieven and B. Scheuermann, "High-speed per-flow traffic measurement with probabilistic multiplicity counting," in *Proc. IEEE INFOCOM*, Mar. 2010, pp. 1–9.

[5] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller, "An overview of IP flow-based intrusion detection," *IEEE Commun. Surveys Tuts.*, vol. 12, no. 3, pp. 343–356, 3rd Quart., 2010.

[6] M. Yoon, T. Li, S. Chen, and J.-K. Peir, "Fit a spread estimator in small memory," in *Proc. IEEE INFOCOM 28th Conf. Comput. Commun.*, Apr. 2009, pp. 504–512.

[7] S. Sen and J. Wang, "Analyzing peer-to-peer traffic across large networks," in *Proc. 2nd ACM SIGCOMM Workshop Internet Measurment*, 2002, pp. 137–150.

[8] J. Anton, L. Jacobs, X. Liu, J. Parker, Z. Zeng, and T. Zhong, "Web caching for database applications with Oracle web cache," in *Proc. ACM SIGMOD Int. Conf. Manag. Data*, 2002, pp. 594–599.

[9] Y. Zhou, Y. Zhang, C. Ma, S. Chen, and O. O. Odegbile, "Generalized sketch families for network traffic measurement," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 3, no. 3, pp. 1–34, Dec. 2019.

[10] A. Akella, A. Bharambe, M. Reiter, and S. Seshan, "Detecting DDoS attacks on ISP networks," in *Proc. 22nd ACM SIGMOD/PODS Workshop Manag. Process. Data Streams*, 2003, pp. 1–3.

[11] J. Mirkovic and P. Reiher, "A taxonomy of DDoS attack and DDoS defense mechanisms," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 2, pp. 39–53, Apr. 2004.

[12] S. Singh, C. Estan, G. Varghese, and S. Savage, "Automated worm fingerprinting," in *Proc. OSDI*, vol. 4, 2004, pp. 1–16.

[13] S. Chen and Y. Tang, "Slowing down internet worms," in *Proc. 24th Int. Conf. Distrib. Comput. Syst.*, 2004, pp. 312–319.

[14] Y. Zhou, Y. Zhou, M. Chen, and S. Chen, "Persistent spread measurement for big network data based on register intersection," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 45, no. 1, p. 67, Sep. 2017.

[15] A. Bronselaer, S. Debergh, D. Van Hyfte, and G. D. Tré, "Estimation of topic cardinality in document collections," in *Proc. SIAM Conf. Data Mining (SDM)*, 2010, pp. 31–39.

[16] S. Heule, M. Nunkesser, and A. Hall, "HyperLogLog in practice: Algorithmic engineering of a state of the art cardinality estimation algorithm," in *Proc. 16th Int. Conf. Extending Database Technol.*, 2013, pp. 683–692.

[17] P. Rob, "Interpreting the data: Parallel analysis with Sawzall," *Sci. Program.*, vol. 13, no. 4, pp. 277–298, 2005.

[18] S. Melnik *et al.*, "Dremel: Interactive analysis of web-scale datasets," *Proc. VLDB Endowment*, vol. 3, pp. 330–339, Sep. 2010.

[19] A. Hall, O. Bachmann, R. Büssow, S. Gănceanu, and M. Nunkesser, "Processing a trillion cells per mouse click," *Proc. VLDB Endowment*, vol. 5, no. 11, pp. 1436–1446, Jul. 2012.

[20] K.-Y. Whang, B. T. Vander-Zanden, and H. M. Taylor, "A linear-time probabilistic counting algorithm for database applications," *ACM Trans. Database Syst.*, vol. 15, no. 2, pp. 208–229, Jun. 1990.

[21] P. Flajolet and G. N. Martin, "Probabilistic counting algorithms for data base applications," *J. Comput. Syst. Sci.*, vol. 31, no. 2, pp. 182–209, Oct. 1985.

[22] H. Wang, C. Ma, S. Chen, and Y. Wang, "Fast and accurate cardinality estimation by self-morphing bitmaps," *IEEE/ACM Trans. Netw.*, early access, Feb. 10, 2022, doi: 10.1109/TNET.2022.3147204.

[23] K. Beyer, P. J. Haas, B. Reinwald, Y. Sismanis, and R. Gemulla, "On synopses for distinct-value estimation under multiset operations," in *Proc. ACM SIGMOD Int. Conf. Manag. Data*, 2007, pp. 199–210.

[24] M. Durand and P. Flajolet, "Loglog counting of large cardinalities," in *Proc. Eur. Symp. Algorithms*. Berlin, Germany: Springer, 2003, pp. 605–617.

[25] P. Flajolet, E. Fusy, O. Gandouet, and F. Meunier, "HyperLogLog: The analysis of a near-optimal cardinality estimation algorithm," in *Proc. Conf. Anal. Algorithm*, Nancy, France, 2007, pp. 137–156.

[26] G. Cormode and S. Muthukrishnan, "Space efficient mining of multigraph streams," in *Proc. 24th ACM SIGMOD-SIGACT-SIGART Symp. Princ. Database Syst.*, 2005, pp. 271–282.

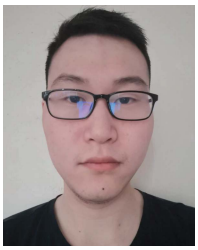
[27] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with opensketch," in *Proc. 10th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2013, pp. 29–42.

[28] Q. Xiao, S. Chen, M. Chen, and Y. Ling, "Hyper-compact virtual estimators for big network data based on register sharing," in *Proc. ACM SIGMETRICS Int. Conf. Meas. Model. Comput. Syst.*, Jun. 2015, pp. 417–428.

[29] G. Cormode and S. Muthukrishnan, "Estimating dominance norms of multiple data streams," in *Proc. Eur. Symp. Algorithms*. Berlin, Germany: Springer, 2003, pp. 148–160.

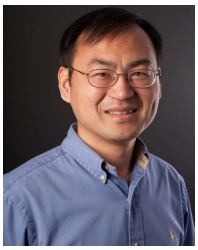
[30] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman, "One sketch to rule them all: Rethinking network flow monitoring with UnivMon," in *Proc. ACM SIGCOMM Conf.*, Aug. 2016, pp. 101–114.
[31] Y. Li, R. Miao, C. Kim, and M. Yu, "FlowRadar: A better NetFlow for data centers," in *Proc. 13th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2016, pp. 311–324.
[32] Q. Huang *et al.*, "SketchVisor: Robust network measurement for software packet processing," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2017, pp. 113–126.
[33] T. Yang *et al.*, "Elastic sketch: Adaptive and fast network-wide measurements," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2018, pp. 561–575.
[34] Z. Liu *et al.*, "Nitrosketch: Robust and general sketch-based monitoring in software switches," in *Proc. ACM Special Interest Group Data Commun.*, Aug. 2019, pp. 334–350.
[35] L. Tang, Q. Huang, and P. Lee, "SpreadSketch: Toward invertible and network-wide detection of superspreaders," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Jul. 2020, pp. 1608–1617.
[36] S. Venkataraman, D. Song, P. B. Gibbons, and A. Blum, "New streaming algorithms for fast detection of superspreaders," School Comput. Sci., Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. CMU-CS-04-142, 2005.
[37] S. Ganguly, M. Garofalakis, R. Rastogi, and K. Sabnani, "Streaming algorithms for robust, real-time detection of DDoS attacks," in *Proc. 27th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, 2007, p. 4.
[38] C. Ma, H. Wang, O. Odegbile, and S. Chen, "Noise measurement and removal for data streaming algorithms with network applications," in *Proc. IFIP Netw. Conf.*, Jun. 2021, pp. 1–9.
[39] C. Ma, H. Wang, O. O. Odegbile, S. Chen, and D. Melissourgos, "Virtual filter for non-duplicate sampling with network applications," *IEEE/ACM Trans. Netw.*, early access, Jun. 22, 2022, doi: 10.1109/TNET.2022.3182694.
[40] M. Moshref, M. Yu, R. Govindan, and A. Vahdat, "SCREAM: Sketch resource allocation for software-defined measurement," in *Proc. 11th ACM Conf. Emerg. Netw. Exp. Technol.*, Dec. 2015, pp. 1–13.
[41] R. B. Basat, X. Chen, G. Einziger, S. L. Feibish, D. Raz, and M. Yu, "Routing oblivious measurement analytics," in *Proc. IFIP Netw. Conf.*, 2020, pp. 449–457.
[42] R. Ben-Basat, G. Einziger, S. L. Feibish, J. Moraney, B. Tayh, and D. Raz, "Routing-oblivious network-wide measurements," *IEEE/ACM Trans. Netw.*, vol. 29, no. 6, pp. 2386–2398, Dec. 2021.
[43] G. Cormode, "Sketch techniques for approximate query processing," in *Foundations and Trends in Databases*. Florham Park, NJ, USA: NOW, 2011.
[44] S. Narayana *et al.*, "Language-directed hardware design for network performance monitoring," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2017, pp. 85–98.
[45] X. Chen, S. Landau-Feibish, M. Braverman, and J. Rexford, "BeauCoup: Answering many network traffic queries, one memory update at a time," in *Proc. Annu. Conf. ACM Special Interest Group Data Commun. Appl., Technol., Archit., Protocols Comput. Commun.*, Jul. 2020, pp. 226–239.
[46] C. Ma, S. Chen, Y. Zhang, Q. Xiao, and O. O. Odegbile, "Super spreader identification using geometric-min filter," *IEEE/ACM Trans. Netw.*, vol. 30, no. 1, pp. 299–312, Feb. 2022.
[47] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen, "Sketch-based change detection: Methods, evaluation, and applications," in *Proc. 3rd ACM SIGCOMM Conf. Internet Meas.*, 2003, pp. 234–247.
[48] UCSD. (2015). *Caida UCSD Anonymized 2015 Internet Traces*. [Online]. Available: https://www.caida.org/data/passive/passive_2015_dataset.xml
[49] Y. Zhou *et al.*, "Cold filter: A meta-framework for faster and more accurate stream processing," in *Proc. Int. Conf. Manag. Data*, 2018, pp. 741–756.

Haibo Wang (Graduate Student Member, IEEE) received the B.E. degree in nuclear science and the master's degree in computer science from the University of Science and Technology of China in 2016 and 2019, respectively. He is currently pursuing the Ph.D. degree with the Department of Computer and Information Science and Engineering, University of Florida. His main research interests include the Internet traffic measurement, software defined networks, and optical circuit scheduling. His work received IEEE ICNP2021 Best Paper Award.



Chaoyi Ma (Graduate Student Member, IEEE) received the B.S. degree in computer information security from the University of Science and Technology of China in 2018. He is currently pursuing the Ph.D. degree in computer and information science and engineering with the University of Florida, under the supervision of Prof. Shigang Chen. His research interests include big data, network traffic measurement, computer network security, and data privacy in machine learning. His work received IEEE ICNP2021 Best Paper Award.

Olufemi O. Odegbile received the B.S. degree in mathematics from the University of Ibadan, Nigeria, the master's degree in computer science from Boston University, USA, and the Ph.D. degree in computer science from the University of Florida. He is currently an Assistant Professor with the Department of Computer Science, Clark University, Worcester, MA, USA. His research interests include computer networks, network security, network traffic measurement, and RFID Technology.



Shigang Chen (Fellow, IEEE) received the B.S. degree in computer science from the University of Science and Technology of China in 1993 and the M.S. and Ph.D. degrees in computer science from the University of Illinois at Urbana-Champaign in 1996 and 1999, respectively. After graduation, he had worked with Cisco Systems for three years before joining the University of Florida in 2002. He is currently a Professor with the Department of Computer and Information Science and Engineering, University of Florida. He held the University of Florida Research Foundation Professorship and the University of Florida Term Professorship. He published over 200 peer-reviewed journal/conference papers. He holds 13 U.S. patents, and many of them were used in software products. His research interests include the Internet of Things, big data, cybersecurity, data privacy, edge-cloud computing, intelligent cyber-transportation systems, and wireless systems. He is an ACM Distinguished Scientist. He received the NSF CAREER Award and other research/best paper awards. He served in various chair positions or as a committee members for numerous conferences. He served as an Associate Editor for IEEE TRANSACTIONS ON MOBILE COMPUTING, IEEE/ACM TRANSACTIONS ON NETWORKING, and a number of other journals.

Jih-Kwon Peir received the master's degree from the University of Wisconsin–Milwaukee in 1981 and the Ph.D. degree in computer science from the University of Illinois in 1986.

He joined the IBM T. J. Watson Research Center and served as a Research Staff Member during 1986 to 1992. At IBM, he participated in the design and development of high-performance mainframe computers. From 1992 to 1994, he joined the Computer and Communication Laboratory, Taiwan, as the Deputy Director of the Computer System Division, where he oversaw the development of an Intel Pentium-based symmetric multiprocessor system. Since 1994, he has been a Faculty Member at the Department of Computer and Information Science and Engineering, University of Florida. He spent a sabbatical year and several summers as a Visiting Professor at the Intel's Microprocessor Research Laboratory and the IBM's Almaden Research Center. He has published over 100 papers in international journals and conferences, book chapters as well as in IBM invention disclosures. He has six patents and nine published inventions. He received an IBM Invention Achievement Award and filed several patents in cache memories. He received two best paper awards at the IEEE International Conference on Computer Design (ICCD) in 1990 and 2001. His paper, "Bloom Filtering Cache Misses for Accurate Data Speculation and Prefetching," was selected to be included in the 25 years of ACM International Conference on Supercomputing (ICS). He received a National Science Foundation Career Award in 1996, and an IBM Research Partnership Award in 1995. He also received an outstanding Alumni Award from the College of Engineering, University of Wisconsin–Milwaukee, in 2010.