Towards Guaranteed Accuracy for Flow Spread Measurement with (ϵ, β)-nonduplicate Sampling

Haibo Wang[1] Chaoyi Ma[2] Dimitrios Melissourgos[3] Guoju Gao[4] Shigang Chen[5]
[1]Department of Computer Science, University of Kentucky, Lexington, KY USA
[2]Google Sunnyvale, CA USA
[3]School of Computing, Grand Valley State University, Allendale, MI, USA
[4]School of Computer Science and Technology, Soochow University, Suzhou China
[5]Department of Computer & Information Science & Engineering, University of Florida, FL USA

***Abstract*—Per-flow spread measurement in high-speed networks is important to many practical applications. To fit in the limited on-chip memory, sketch-based solutions allow multiple flows to share space, causing inter-flow noise and thus sacrificing in accuracy. Recent progress on non-duplicate sampling creates a new direction of sampling-based solutions for spread estimation, which performs better than memory-sharing sketches. However, the current sampling-based solutions either use a system-wide sampling probability or lack the flexibility of setting the sampling probability dynamically and at per-flow level. This paper advances the theory and design of non-duplicate sampling by introducing a new (ϵ, β)-RE accuracy model for spread estimation and a new (ϵ, β)-nonduplicate sampling type, establishing their equivalency, proposing the idea of individualized per-flow sampling, and designing a novel algorithm based on this idea to implement (ϵ, β)-nonduplicate sampling and thus achieving (ϵ, β)-RE accuracy. Trace-driven experiments demonstrate that our new solution outperforms the best state of the art significantly in terms of maximum supported packet stream size under a given accuracy requirement or in terms of accuracy with the same packet stream size, and outperforms sketch-based solutions to spread estimation significantly in accuracy.**

I. Introduction

Per-flow spread measurement over packet streams in a network is a fundamental task for a wide range of applications such as traffic engineering, anomaly detection, access profiling, content caching and so on [1], [2], [3], [4], [5], [6], [7]. Flow spread is defined as the number of *distinct* elements in a flow, where the definition of flow and element can be flexibly configured to serve applications with different measurement requirements. For instance, to detect DDoS attacks [3], [5], [7], we may define a per-destination flow as all packets sent to the same detection address and measure its spread by counting the number of distinct sources (elements) that contact this destination. As another example, we may treat the destination addresses as elements and measure a per-source flow's spread, which helps to find out the super spreaders [8], [9].

Performing per-flow spread measurement at high-speed network devices is challenging due to the space and processing speed constraints on data-plane network processors. On the one hand, it takes bitmap [10], FM sketch [11], or HyperLogLog sketch [12], [13] hundreds or thousands of bits to measure the spread of a single flow. The reason is that counting distinct elements that appear in a flow requires us to remember the elements for duplicate filtering. On the other hand, a modern high-speed router may face millions of concurrent flows. Hence, maintaining a separate data structure per flow requires an unaffordable amount of on-chip memory (e.g., SRAM) from network processors, which must commit such resource to other key network functions such as routing, packet scheduling, traffic shaping and cybersecurity.

Prior art and limitations. There are broadly two categories of solutions to the constraint of memory allocation: *memory-sharing sketches*[1] and *sampling*. Memory-sharing sketches [14], [15], [16], [17], [18], [19] let all flows share memory space. This allows them to fit in a pre-allocated, limited space, achieving memory efficiency at the cost of accuracy in their spread estimation. Thanks to space sharing, the spread information of a flow recorded in a sketch becomes noise to other flows. A major challenge is how to remove inter-flow noise to improve the quality of spread estimation.

Other solutions are based on non-duplicate sampling [20], [21], [22], [23], [24], [25]. Unlike traditional sampling methods that sample each packet independently with a certain probability, non-duplicate sampling methods only sample each packet at its first appearance with a certain probability, and ignore the subsequent appearances of the same packet, called duplicates. In the context of this paper, we abstract each packet as a value pair consisting of a flow label and an element of measurement interest, both of which are extracted from the packet headers or sometimes the payload. Two packets are considered to be the same if they share the same flow label and the same element. With non-duplicate sampling, the sampled packets no longer have duplicates. We can use a counter for each flow to keep track of the number of sampled packets from that flow and derive the flow spread by taking the sampling probability into consideration. The experiments in [22], [23], [24] show that the non-duplicate sampling solutions outperform the memory-sharing sketches with better accuracy and smaller overhead in spread estimation. Moreover, they keep the identifiers of the sampled flows, whereas many memory-sharing sketches do not. This paper focuses on advancing the theory and design of novel non-duplicate sampling solutions.

The existing non-duplicate sampling solutions can be further divided into two sub-categories: *uniform non-duplicate sampling* (UNS) [22], [23], [24] and *non-uniform nonduplicate sampling* (NNS) [20], [21], [25]. UNS samples each packet at its first appearance with a preset, fixed probability. NNS allows the sampling probability to evolve in a specific way that cannot be controlled or set dynamically in real time. We observe that

[1]Sketches are compact data structures.

such inflexibility in sampling probability restricts the room for UNS/NNS to improve their performance. Moreover, there lacks a rigorous accuracy model for spread estimation in most non-duplicate sampling solutions.

Our contribution. This paper advances the state of the art in non-duplicate sampling research with the following contributions.

- We formally introduce the (ϵ, β)-RE accuracy model for per-flow spread estimation based on the relative root-mean-square error, which captures both the bias and the variance in the estimates. We formalize per-flow spread estimation as a constrained optimization problem: Given an (ϵ, β)-RE accuracy requirement with user-specified parameters ϵ and β, our goal is to maximize the size of the packet stream that we can handle with a certain amount of memory allocation.
- We introduce a new type of sampling, called (ϵ, β)-nonduplicate sampling, and establish the equivalency between implementing (ϵ, β)-nonduplicate sampling and achieving (ϵ, β)-RE accuracy. This establishes the theoretical foundation of ensuring spread estimation accuracy in the sampling solutions.
- We prove two theorems that lay the ground for modifying the existing UNS and NNS solutions to perform (ϵ, β)-nonduplicate sampling. We observe that the existing work lacks the flexibility in selecting per-flow sampling probabilities, which limits their efficiency in supporting (ϵ, β)-nonduplicate sampling.
- We propose the idea of individualized non-duplicate sampling (INS), which allows us to set each flow's sampling probability dynamically and independently from other flows. We design an algorithm based on the idea of INS to implement (ϵ, β)-nonduplicate sampling more efficiently than the existing UNS and NNS. We prove the correctness of the algorithm and determine its optimal parameter.
- We perform extensive experiments using real Internet traces to evaluate our INS-based solution for spread estimation in comparison with the state of the art. The experimental results show that the new solution outperforms the alternatives based on the existing UNS and NNS. It can support much larger packet streams under the (ϵ, β)-RE accuracy requirement with the same memory allocation. When dealing with packet streams of the same size, the new solution achieves much better accuracy in spread estimation.

II. Preliminaries

A. Flow Model

Consider a packet stream received by a router, a firewall, or a network device of other type. Each packet is abstracted as a pair, $\langle f, e\rangle$, where f is a flow label and e is an element, which are both extracted from the packet based on application need. Label f may be source address, destination address, or any address/port/flag combination (such as the TCP identifier), found in the packet headers. All packets carrying the same label f form a flow f. Element e is a value or a value combination of application interest in the packet headers or the payload other than the flow label. The *spread* of flow f is defined as the number of *distinct* elements in the flow during a measurement period. This statistic is important to a wide range of applications. Consider per-source flows where the source address of each packet is the flow label and the destination address is the element under measurement. The spread of each flow, i.e., the number of distinct destinations that each source contacts during a measurement period, helps us track network reconnaissance activities, worm-infected hosts, botnet communications, and malicious scanners [26], [27]. Consider per-destination flows where the destination address is the flow label and the source address is the element of interest. The spread of each flow, which is the number of distinct sources that contacts each destination, helps us track potential botnet-based denial-of-service or denial-of-quality attacks, service hotspots, or congested network activities [5], [28].

B. Spread Estimation with (ϵ, β)-RE Accuracy

Measuring the spread of a flow precisely requires us to remember all elements that we have seen so far, which can be costly as flow spreads can be in millions or more. This study considers how to estimate flow spread with a space-efficient, probabilistic measurement approach. For an arbitrary flow f, let s_f be its actual spread and $\hat{s}_f$ be an estimate, which is a random variable under a probabilistic measurement approach [20], [22], [23], [25]. To assess the accuracy of the estimation, we use the *relative root-mean-square error* (RRMSE or RE) [29], [30], [31], which is defined below.

$$\text{RE}(\hat{s}_f) = \sqrt{\text{E}[(\frac{\hat{s}_f}{s_f} - 1)^2]}, \tag{1}$$

where $\text{E}[\cdot]$ is the expectation of a random variable. $\text{RE}(\hat{s}_f)$ captures both the bias and the variance of the estimation due to

$$\begin{aligned}[\text{RE}(\hat{s}_f)]^2 &= \frac{(\text{E}[\hat{s}_f] - s_f)^2 + \text{E}[(\hat{s}_f - \text{E}[\hat{s}_f])^2]}{{s_f}^2} \\ &= \frac{[\text{Bias}(\hat{s}_f)]^2 + \text{Var}(\hat{s}_f)}{{s_f}^2}.\end{aligned}$$

Refer to [32] for detailed proof.

We define an (ϵ, β)-RE accuracy requirement as follows: For any flow with $s_f \geq \beta$, its relative root-mean-square error (RE) must be bounded by ϵ, i.e.,

$$\text{RE}(\hat{s}_f) \leq \epsilon, \tag{2}$$

where $\epsilon \in (0, 1)$ and $\beta \geq 1$ are parameters that are set by the user based on application need. As an example in the introduction, considering a packet stream in a network, if we set flow label as the source address and element as the destination address in packet header, each flow consists of all packets from a common source and its spread is the number of distinct destination addresses that the source has contacted. If the spread of the flow from a source is greater than a threshold β, we consider the source as a potential scanner that requires further monitoring or countering actions. The value of β will depend on traffic statistics of the network. If the spreads from most normal hosts (sources) are below 10, we can set β to 10; but if most normal spreads are below 50, we will set β accordingly. Similarly, the value of ϵ is set according to user requirement, such as 5%, and system tradeoff. There is tradeoff between accuracy and memory requirement, as a smaller value of ϵ means a larger amount of memory needed (Theorem 6).

C. Maximizing Stream Size

The accuracy requirement of (2) serves as a constraint in our spread estimation design. Any data structure for keeping track of flow spread must have a way to record the elements that have been seen in order to remove duplicates. Given a certain amount of memory allocation, as more and more new elements are recorded, the data structure will become saturated and will no longer be able to meet (2). In that case, we have to stop the current measurement, store the results in a server, clear the data structure, and start the next run of measurement. Therefore, any design will have a limit in its supported stream size over one continuous run of measurement, denoted as n, which is the number of distinct elements from all flows in the stream that are processed before the data structure saturates. Note that elements of different flows are considered to be distinct, such as $\langle f, e\rangle$ and $\langle g, e\rangle$. *The goal of this paper is to design a novel and efficient spread estimation solution that maximizes its supported stream size under a given (ϵ, β)-RE accuracy requirement and a given amount of memory.*

III. (ϵ, β)-Nonduplicate Sampling

To facilitate the implementation of (ϵ, β)-RE accuracy for spread estimation, we introduce a new type of sampling, called (ϵ, β)-*nonduplicate sampling*. It samples each distinct packet at its first appearance in a flow with a certain probability (which may be set as a constant or change over time), and ignores all subsequent appearances. Before giving the formal definition of (ϵ, β)-nonduplicate sampling, we define some notations: Consider an arbitrary flow f in a data stream. Let S_f be the sequence of distinct packets, $\langle f, e\rangle$, in flow f, with $s_f = |S_f|$ being the total number of distinct packets in the flow. Let S'_f be the subsequence of S_f that are sampled, with $s'_f = |S'_f|$ being the total number of sampled packets. Clearly, $S'_f \subseteq S_f$. With probabilistic sampling, s'_f is a random variable. For the kth packet in S'_f where $1 \le k \le s'_f$, let m_k be the index of the same packet in S_f. Under probabilistic sampling, m_k is also a random variable, but in a specific run like the example below, it has an instance value.

Suppose flow f has three packets, $\langle f, e\rangle$, $\langle f, e\rangle$, $\langle f, e'\rangle$. S_f is the sequence of two distinct packets, $\langle f, e\rangle, \langle f, e'\rangle$. Suppose it is sampled with 50% probability and S'_f contains only $\langle f, e'\rangle$. Then m_1 = 2, i.e., the first packet ($k = 1$) in S'_f has an index of 2 (or is the second packet) in S_f.

For the last sampled packet in S'_f, i.e., $k = s'_f$, we denote $m_{s'_f}$ as m^*_f for simplicity. This is an important value. Later, our solution will be designed to measure the instance value of s'_f in a sampling run, compute $E[m^*_f]$ from the measured value of s'_f, and use $E[m^*_f]$ as our spread estimate for flow f, i.e., $\hat{s}_f = E[m^*_f]$.

Definition 1: (ϵ, β)-*nonduplicate sampling* only samples a packet at its first appearance with a certain probability and for any flow f with $s_f \ge \beta$, it ensures that

$$\mathrm{RE}(m^*_f) \le \epsilon, \tag{3}$$

where

$$\mathrm{RE}(m^*_f) = \sqrt{\mathrm{E}[\frac{m^*_f}{\mathrm{E}[m^*_f]} - 1]^2}. \tag{4}$$

The implementation of (ϵ, β)-nonduplicate sampling is to find a way to determine the sampling probability such that (3) can be ensured. We have the following theorem that establishes the linkage between (ϵ, β)-nonduplicate sampling and spread estimation with (ϵ, β)-RE accuracy.

Theorem 1: Given a solution that performs (ϵ, β)-*nonduplicate sampling*, by setting $\hat{s}_f = E[m^*_f]$, we have

$$\mathrm{E}(\hat{s}_f) = s_f$$
$$RE(\hat{s}_f) \le \epsilon\sqrt{\frac{1}{1-\epsilon^2}}$$

On the one hand, $E[m^*_f]$, i.e., $E[m_{s'_f}]$, is a fixed value under a specific sampling run that results in a specific instance value of s'_f. Note that $m_{s'_f}$ is a random variable under any fixed value of s'_f. On the other hand, s'_f is a random variable whose value changes under different sampling runs, and thus $E[m_{s'_f}]$, i.e., the flow spread estimate $\hat{s}_f$, is also a random variable (related to s'_f) under different runs. The above theorem gives the expected value and the error of the estimate $\hat{s}_f$ under all possible runs.

The proof is provided in the Appendix. When ϵ is small, $\sqrt{\frac{1}{1-\epsilon^2}} \approx 1$. If we want $RE(\hat{s}_f) = \epsilon$, we have to slightly tighten sampling to $(\epsilon\sqrt{\frac{1}{1+\epsilon^2}}, \beta)$-nonduplicate sampling, which results in the following corollary.

Corollary 1: A solution that performs $(\epsilon\sqrt{\frac{1}{1+\epsilon^2}}, \beta)$-*nonduplicate sampling* and computes $E[m^*_f]$ for any flow f with $s_f \ge \beta$ ensures (ϵ, β)-RE accuracy if we set $\hat{s}_f = E[m^*_f]$.

Our task now becomes how to efficiently solve (ϵ, β)-nonduplicate sampling, measure s'_f and compute $E(m^*_f)$. Note that in the actual implementation we will use $(\epsilon\sqrt{\frac{1}{1+\epsilon^2}}, \beta)$-nonduplicate sampling.

IV. Individualized Non-duplicate Sampling

This is the first paper that introduces the problem of (ϵ, β)-nonduplicate sampling. Below we will adapt some existing solutions for other nonduplicate sampling types to address this problem. We will then point out their inefficiency, and introduce our new idea to solve the problem efficiently.

A. Uniform Non-duplicate Sampling (UNS)

There exists prior work [22], [23], [24] that performs *uniform non-duplicate sampling* (UNS), which samples each packet $\langle f, e\rangle$ with a fixed, preset sampling probability p at the packet's first appearance and ignores the duplicates. But they treat p as a user input. Without knowing how to set p automatically and properly, they cannot solve the problem of (ϵ, β)-nonduplicate sampling. As one of the major contributions in this paper, we figure out how to set p for (ϵ, β)-nonduplicate sampling in the following theorem. (All proofs can be found in the Appendix.)

Theorem 2: Any uniform non-duplicate sampling solution with $p \ge \frac{1}{1+\epsilon^2\beta}$ implements (ϵ, β)-nonduplicate sampling.

Any UNS solution will need a data structure (typically a bitmap) to record distinct elements that have been sampled so far [20], [21], [22], [23], [24], [25]. We want to keep the sampling probability p as small as possible. Intuitively, smaller p reduces the number of sampled packets that need to be recorded, allowing a larger data stream to be recorded before the data structure is saturated under the constraint of

(2). The smallest sampling probability for any UNS solution is $p = \frac{1}{1+\epsilon^2\beta}$. We denote this probability as p_β. The UNS solutions use the same, fixed sampling probability for all flows. Next we argue that using individualized, dynamic sampling probabilities can do better.

B. Our Idea: Individualized Nonduplicate Sampling (INS)

We have proved the following theorem as the theoretical foundation for implementing (ϵ, β)-nonduplicate sampling with individualized sampling probabilities, one for each flow in a data stream.

Theorem 3: Any non-duplicate sampling solution with an individualized sampling probability $p_f \geq \min\{\frac{1}{1+\epsilon^2 s_f}, \frac{1}{1+\epsilon^2\beta}\}$ for each flow f implements (ϵ, β)-nonduplicate sampling.

For very large flows with $s_f \gg \beta$, which often contribute most of distinct elements to the data stream, p_f is much smaller than p_β. In fact, p_f approaches to zero as s_f increases. Therefore, using individualized sampling probabilities as suggested by Theorem 3 can greatly reduce the number of sampled packets and thus increase the data stream size that we can handle. The UNS solutions cannot do that by definition, due to their uniform (fixed) sampling probability. In fact, Theorem 2 uses the most conservative sampling probability for all flows.

For better performance, according to Theorem 3, we need to control the sampling probabilities of individual flows and properly set them to different values. That is a capability absent from all existing work, include UNS [22], [23], [24] and non-uniform nonduplicate sampling (NNS) [20], [21]. Both UNS and NNS uses a single sampling probability p for all flows, but NNS allows p to decrease as the data structure is more and more saturated. To support (ϵ, β)-nonduplicate sampling, this probability p must stay above p_β. In contrast, our individualized nonduplicate sampling (INS) proposed in this paper is different: Each flow has its own sampling probability, which can be set dynamically and is independent of other flows. Unlike [20], [21], the sampling probability of a flow does not decrease as the data structure records packets of *other flows*. Most importantly, we need a way to determine the proper value that each flow's individual sampling probability should be set to, and do so dynamically as the packets of the flow are recorded, not only to implement (ϵ, β)-nonduplicate sampling, but also to maximize the stream size that we can support.

Ideally, for each flow f, we would simply set $p_f = \frac{1}{1+\epsilon^2 s_f}$ for the optimal result. But we do not know s_f, which is what we want to estimate. Our idea is: We begin with $p_f = p_\beta = \frac{1}{1+\epsilon^2\beta}$. As we sample packets from flow f and record them, we estimate the number of distinct elements in flow f that we have seen so far. When this estimate exceeds β, we will be able to use the current estimate to replace s_f in the formula of p_f to produce a sampling probability that is smaller than p_β. As we record more and more packets from flow f, the value of p_f will be continuously decreased towards the optimal value of $\frac{1}{1+\epsilon^2 s_f}$.

C. Algorithmic Design of INS

Consider an arbitrary flow f. Let $p_f(k)$ be the sampling probability that we will set after k packets in f are sampled and recorded. When $k \leq \beta \times p_\beta$, we keep $p_f(k) = p_\beta$. The reason is that, as long as $k \leq \beta \times p_\beta$, with a fixed sampling probability p_β, the estimated number of distinct elements in flow f that we have seen is $k\frac{1}{p_\beta} \leq \beta$. But when $k > \beta \times p_\beta$, that estimated number will be larger than β and we will need to set $p_f(k)$ smaller than p_β. The exact value of $p_f(k)$ is given by the following theorem. To simplify the notation, let $\bar{k} = \lceil \beta \times p_\beta \rceil$.

Theorem 4: Any non-duplicate sampling solution implements (ϵ, β)-nonduplicate sampling if it sets an individualized sampling probability $p_f(k)$ for each flow f as follows: For $0 \leq k < \bar{k}$, $p_f(k) = p_\beta$. For $k \geq \bar{k}$,

$$p_f(k) = \frac{1-\epsilon^2}{2\mathrm{E}[m_k]\epsilon^2 + 1}, \tag{5}$$

where

$$\mathrm{E}[m_k] = \begin{cases} (1+\epsilon^2\beta)k; & 0 < k \leq \bar{k} \\ (\frac{1+\epsilon^2}{1-\epsilon^2})^{k-\bar{k}}\mathrm{E}[m_{\bar{k}}] + \frac{1-(\frac{1+\epsilon^2}{1-\epsilon^2})^{k-\bar{k}}}{-2\epsilon^2}; & k > \bar{k}. \end{cases} \tag{6}$$

The proof can be found in the Appendix. Refer to Section III for the definition of m_k. According to Corollary 1, we can use (6) as the estimation formula for $\hat{s}_f$: At the end of the measurement, $k = s'_f$ and we compute $\hat{s}_f = \mathrm{E}[m^*_f] = \mathrm{E}[m_{s'_f}]$ from (6). Similarly, at any time, we can compute the spread of flow f thus far, which is $\mathrm{E}[m_k]$, from (6).

With the individualized sampling probabilities given by Theorem 4, we describe our algorithm for non-duplicate sampling. The main data structure is a bitmap B of m bits, which are initialized to zeros. Let $B[i]$, $0 \leq i < m$, be the ith bit in B. We also maintain a HashMap M. Each sampled flow f has a key-value entry in M, with the key being the flow label f and the value being a counter c_f for spread estimation. In our case, c_f keeps track of the number of sampled packets from flow f. M is common overhead for any sampling-based solution for spread measurement.

For each arrival packet $\langle f, e \rangle$, we take the two steps of operation. The first step performs sampling at a variable probability p_1. The second step performs duplicate filtering that has an intrinsic sampling component with a fixed sampling probability $p_2 = \max\{p_\beta, \frac{1}{e}\}$. This choice of p_2 is to achieve optimal performance, as we will prove later. The combined sampling probability $p_1 p_2$ should be $p_f(c_f)$, as defined in (5). Hence, we should set $p_1 = \frac{p_f(c_f)}{p_2}$ in the first step. If the packet is sampled successfully by both steps, we increase c_f by one. Operation details are given below.

Step 1: *Sampling.* Retrieve c_f from M. If there is no entry of f in M, let $c_f = 0$. Compute $p_f(c_f)$ from (5). Let $p_1 = \frac{p_f(c_f)}{p_2} = \frac{p_f(c_f)}{\max\{p_\beta, \frac{1}{e}\}}$. Since $p_f(c_f)$ will only decrease as more packets from f are sampled and recorded by c_f, p_1 *will only decrease as well*, which is an important property for the correctness proof later. We perform sampling with probability p_1 in this step. It can be done by computing a hash value $h(f, e) \in [0, 1)$ where $h(\cdot)$ is a uniform hash function. If $h(f, e) < p_1$, we continue to Step 2; otherwise, the packet is ignored (not sampled).

Step 2: *Duplicate filtering.* We hash the packet $\langle f, e \rangle$ to a bit in B, i.e., $B[H(f, e)]$, where $H(\cdot) \in [0, m)$ is a uniform hash function.

- *Case 1: If* $B[H(f, e)] = 0$, this is the first appearance of the packet. We set $B[H(f, e)] = 1$. We then perform sampling with probability $\frac{mp_2}{z}$, where z is the number of

Algorithm 1: INS for (ϵ, β)-nonduplicate sampling

Input: ϵ, β

Action: Perform (ϵ, β)-nonduplicate sampling

create a HashMap M
create a bitmap B of m bits
set $z = m$, $p_\beta = \frac{1}{1+\epsilon^2\beta}$, $p_2 = \max\{p_\beta, \frac{1}{e}\}$
for *each packet* $\langle f, e\rangle$ **do**
    Step 1: Sampling
    $c_f = M.getDefaultValue(f, 0)$
    $p_1 = \frac{p_f(c_f)}{p_2}$
    generate a hash value $r = h(f, e) \in [0, 1)$
    if $r < p_1$ **then**
        Step 2: Duplicate filtering
        if $B[H(f, e)] = 0$ **then**
            if $H(f, e) < \frac{m^2 p_2}{z}$ **then**
                $M.put(f, c_f + 1)$ `// the packet is sampled`
            set $B[H(f, e)] = 1$
            $z = z - 1$
            if $z < mp_2$ **then**
                terminate sampling

zeros in B and m is the length of B. Only if the packet is sampled, we record the packet by increasing c_f by one and write c_f back to M.

- *Case 2: If* $B[H(f, e)] = 1$*,* we ignore the packet to avoid duplicate.

It is possible that even though this is the first appearance of $\langle f, e\rangle$, we hit Case 2 because $B[H(f, e)]$ may have been set to one by another packet due to hash collision. Nonetheless, the probability for hitting Case 1 is $\frac{z}{m}$, and then the probability of being sampled in Case 1 is $\frac{mp_2}{z}$. Hence, the combined probability for a packet to be recorded through Case 1 is $\frac{z}{m} \times \frac{mp_2}{z} = p_2$ at its first appearance. For all future appearances, the packet will hit Case 2 because $B[H(f, e)]$ is set to 1 after the first appearance.

The above algorithm is formally described in Alg. 1. Step 1 is designed to customize the sampling probability of each flow individually through the use of p_1. Different flows may have different numbers of distinct packets recorded (i.e.,c_f) and consequently their sampling probabilities will be set differently. Step 2 uses B to sample each packet at its first appearance with probability p_2 and to filter duplicates. Keeping p_2 a constant in this step ensures that p_1 can only decrease in Step 1, which is needed in our correctness proof.

The algorithm terminates when $z < mp_2$. Initially, the number of zeros in B is m and it will decrease as bits are set to ones, which in turn will increase the sampling probability $\frac{mp_2}{z}$ in Case 1 of Step 2. When $z < mp_2$, that sampling probability becomes greater than one, which is not possible.

The following three theorems establish the correctness, the optimality of Alg. 1 and the minimum memory allocation needed. Their proofs can be found in Appendix.

Theorem 5 (Correctness): Alg. 1 will sample and record any packet $\langle f, e\rangle$ with probability $p_f(c_f)$ at its first appearance and ignore the packet's subsequent appearances, where c_f is the number of sampled packets from flow f at the time when packet $\langle f, e\rangle$ first arrives.

Theorem 6 (Optimal parameter setting for p_2): The optimal value of p_2 that maximizes the expected number of distinct packets that Alg. 1 can handle is

$$p_2 = \max\{p_\beta, \frac{1}{e}\}. \tag{7}$$

V. Evaluation

We evaluate the performance of the proposed solutions on the software platform through experiments based on real-world data traces.

A. Experimental Setting

There is no prior work for either spread estimation with (ϵ, β)-RE accuracy or (ϵ, β)-nonduplicate sampling. We implement, evaluate and compare three solutions developed in this paper: (1) the INS solution; (2) a solution, denoted as VF+, designed from our Theorem 2 based on the best uniform nonduplicate sampling algorithm, VF [24], with a stream-wide fixed sampling probability p_β; (3) a solution, denoted as FreeRS+, designed from our Theorem 3 based on the best non-uniform nonduplicate sampling algorithm, FreeRS [21], with a stream-wide sampling probability that decreases from 1 to p_β. We also include an existing work SAS-LC [25]. Although it also adapts sampling probabilities of different flows, it has no mechanism to control these probabilities and therefore it cannot achieve (ϵ, β)-RE accuracy or (ϵ, β)-nonduplicate sampling. We implement it for some of the comparisons. Paper [25] also proposes another sampling solution called SAS-LOG, which is not compared in this paper because the experiments in [25] have already demonstrated that it is inferior to SAS-LC for flow spread measurement in terms of accuracy and memory efficiency (but has advantage for another task called flow size measurement).

Our experimental comparison is based on the following three performance metrics. 1) *Maximum supported stream size*, as defined in Section II-B. Given an (ϵ, β)-RE accuracy requirement and a certain memory allocation, our goal is to maximize the stream size (in terms of the number of *distinct* packets in the stream) that we can handle. For this, we compare INS, VF+ and FreeRS+, but not SAS-LC because it does not support (ϵ, β)-RE accuracy. 2) *Processing time*, which is the average time it takes to process a packet. Processing time can be translated into streaming throughput. An average processing time of 100ns corresponds to a throughput of 10 million packets per second. For a packet stream, that is equivalent to 10Gbps if the average packet size is 1k bits. 3) *Accuracy.* To include SAS-LC in the comparison, we drop the constraint of (ϵ, β)-RE accuracy, but feed the same data stream that INS can handle to other solutions. In this case, we can only use the original work of VF and FreeRS, which can handle larger streams without being limited by the constraint. Then we measure and compare the percentages of flows that can still meet (2).

The dataset used in our evaluation is real Internet traffic traces downloaded from CAIDA [33]. The trace has 20 mins of packets and contain 400 million packets. In our experiments, the flow labels are the destination addresses in packet IP header

TABLE I: Maximum supported stream size of FreeRS+, VF+ and INS under different values of ϵ and different memory allocations (Mbits) over the normal dataset. β = 5.

ϵ	0.2			0.15			0.1		
Mem.	FreeRS+	VF+	INS	FreeRS+	VF+	INS	FreeRS+	VF+	INS
12.8	726111	2345103	4860917	416609	1366639	2648289	188933	627092	1020194
6.4	359676	1168832	2466170	208215	683269	1225735	94416	312534	493710
3.2	180165	584983	1095587	104049	341228	581010	47078	156365	236300
1.6	90064	291935	516507	52117	170532	280912	23628	78084	108986
0.8	45000	146135	248878	26021	85319	130585	11722	39026	49883

TABLE II: Maximum supported stream size of FreeRS+, VF+, and INS under different values of ϵ and different memory allocations over the large dataset. $\beta = 100$.

ϵ	0.2			0.15			0.1		
Mem(Mbits)	FreeRS+	VF+	INS	FreeRS+	VF+	INS	FreeRS+	VF+	INS
51.2	38125464	99776071	457800134	24045818	64844082	286347902	12500002	37034447	138674464
25.6	19120122	49654270	245434736	12036480	31827573	125922932	6258780	18428144	57151121
12.8	9550810	24436420	92957055	6022274	15903406	51097561	3129011	9199036	24169101
6.4	4791819	12202811	42860077	3004795	7948300	23977130	1547925	4594863	12918407
3.2	2386958	6119590	22007084	1490957	3947593	14622598	762617	2298988	5665151

TABLE III: Maximum supported stream size of FreeRS+, VF+ and INS under different values of ϵ and different memory allocations over the skewed dataset. β = 5.

ϵ	0.2			0.1		
Mem(Mbits)	FreeRS+	VF+	INS	FreeRS+	VF+	INS
12.8	726111	2345103	4849501	188933	627092	1224526
6.4	359676	1168832	2489987	94416	312534	772735
3.2	180165	584983	1285128	47078	156365	557573
1.6	90064	291935	805219	23628	78084	453670
0.8	45000	146135	568237	11722	39026	404600

TABLE IV: Percentage of flows in each bin with their empirical REs bounded by ϵ, over the first 400k distinct packets of the normal dataset. Memory allocated is 6.4Mbits, $\beta = 5$ and $\epsilon = 0.1$.

Bin	$[5,2^3]$	$(2^3,2^4]$	$(2^4,2^5]$	$(2^5,2^6]$	$(2^6,2^7]$	$(2^7,2^8]$	$(2^8,2^9]$	$(2^9,2^{10}]$	$> 2^{10}$
Flow No.	5580	2171	799	437	178	110	66	19	22
FreeRS	10.8%	48.1%	98.8%	100%	100%	100%	100%	100%	100%
VF	0	0	0	78.7%	100%	100%	100%	100%	100%
SAS-LC	0	0	2.9%	74.8%	100%	100%	100%	100%	100%
INS	96.4%	100%	100%	100%	100%	100%	100%	100%	100%

TABLE V: Percentage of flows in each bin with their empirical REs bounded by ϵ, over the first 750k distinct packets of the skewed dataset. Memory allocated is 6.4Mbits, $\beta = 5$ and $\epsilon = 0.1$.

Bin	$[5,2^3]$	$(2^3,2^4]$	$(2^4,2^5]$	$(2^5,2^6]$	$(2^6,2^7]$	$(2^7,2^8]$	$(2^8,2^9]$	$> 2^9$
No. of flows	6736	2654	926	518	157	143	73	50
FreeRS	0	0	24.1%	98.2%	100%	100%	100%	100%
VF	0	0	2.1%	80.4%	100%	100%	100%	100%
SAS-LC	0	0	3.0%	79.8%	100%	100%	100%	100%
INS	96.8%	100%	100%	100%	100%	100%	100%	100%

TABLE VI: Percentage of flows in each bin with their empirical REs bounded by ϵ, over the first 3M distinct packets of the skewed dataset. Memory allocated is 6.4Mbits, $\beta = 32$ and $\epsilon = 0.1$.

Bin	$[2^5,2^6]$	$(2^6,2^7]$	$(2^7,2^8]$	$(2^8,2^9]$	$(2^9,2^{16}]$	$> 2^{16}$
No. of flows	5941	2234	764	292	321	5
FreeRS	5.7%	65.1%	95.8%	100%	100%	100%
VF	8.3%	29.2%	34.5%	34.2%	42.3%	100%
SAS-LC	35.8%	61.9%	77.6%	75.0%	85.0%	100%
INS	99.6%	100%	100%	100%	100%	100%

and the elements are the source IP addresses. That is, packets to the same destination form a flow, and only packets with both the same source and the same destination are considered as duplicates. Flow spread is defined as the number of distinct sources that contact the same destination host, which has an application for detecting DDoS attacks. The traces contain 4.9 million distinct packets.

We refer to the above dataset as the *normal dataset*. We create two more datasets from the normal dataset. The first one is called the *skewed dataset*, where the packets of large flows are concentrated in the packet stream. It simulates the scenario of launching malicious attacks where a large number of bots create flush crowd in a short time towards a selected set of targets. To create this dataset, we move the top 10 largest flows with a total spread of 363943 to the front of the traffic trace. The second one is called the *large dataset*, which is produced by expanding the normal dataset by 100 times. For each distinct packet in the normal dataset, we create x distinct packets of the same flow, where x is a random number from $[1, 200]$. Our experiments are performed on a computer with Inter Core Xeon W-2135 3.7GHz and 32 GB memory.

B. Results of Maximum Supported Stream Size

Our first experiment compares the maximum supported stream sizes of INS, VF+ and FreeRS+ under the constraint of (ϵ, β)-RE accuracy, over the normal/skewed/large datasets. The value of ϵ varies from 0.1 to 0.2 for all datasets. For the normal/skewed datasets, the memory allocation varies from 0.8Mbits to 12.8Mbits and $\beta = 5$. As we have explained in Section IV-C, HashMap is common overhead for any sampling-based solution for spread measurement, its memory is not counted in experiments for each algorithm. For the large dataset, the memory allocated and the value of β are correspondingly increased: memory varies from 3.2Mbits to 51.2Mbits and $\beta = 100$.

When we feed the packets of a dataset to a solution, we stop when its data structure has recorded too many packets and can no longer ensure the (ϵ, β)-RE accuracy. For VF+, that happens when it can no longer perform sampling with probability p_β, a detectable condition in [24], For FreeRS+, that happens when its stream-wide sampling probability drops below p_β. For INS, that happens when the sampling probability in Case 1 of Step 2 is

greater than 100%. When that happens, we have the maximum stream size supported by the solution.

Table I presents the maximum stream sizes supported by different solutions over the normal dataset under different ϵ values (columns) and different memory allocations (rows). INS always supports a larger stream under the same ϵ and memory allocation. For example, When memory allocated is 12.8Mbits and ϵ = 0.2, INS can support 2.07 times and 6.69 times of the stream sizes that VF+ and FreeRS+ can support, respectively, thanks to individualized sampling probabilities.

Our improvement is more significant under the skewed dataset, shown in Table III. INS increases the maximum supported stream size by up to 9.36 and 32.51 compared to VF+ and FreeRS+, respectively. The maximum supported stream sizes of VF+ and FreeRS+ are insensitive to data skewness because their sampling probabilities are the same to different flows. However, INS sets different sampling probabilities to different flows. When the packets of a large flow are concentrated, they will quickly drive the flow's sampling probability down and reduce the number of sampled packets that need to be recorded, which delays the saturation of the underlying data structure.

Table II shows the maximum stream sizes over the large dataset. INS's advantage over FreeRS+ and VF+ remains significant. For instance, when $\epsilon = 0.2$ and $M = 51.2$Mbits, INS extends the maximum supported stream size by 3.58 times and 11.00 times compared to VF+ and FreeRS+, respectively.

C. Processing Time

TABLE VII: Average processing time of FreeRS+, VF+ and INS. Memory allocation is 6.4Mbits, ϵ = 0.1 and $\beta = 5$ for each solution.

Algorithm	FreeRS+	VF+	INS
Time (ns)	136	94	113

Our second experiment compares the average processing time per packet of INS, VF+ and FreeRS+ with $\epsilon = 0.1$, $\beta = 5$, and memory allocation of 6.4Mbits. We feed each solution a portion of the normal dataset up to the maximum size that the solution can support under those parameters, which can be found in Table I. Table VII presents the average processing time. The results show that the processing times of the three solutions are comparable, with VF+ modestly better than INS, which is in turn modestly better than FreeRS+. They can all process millions of packets per second.

D. Results of RE

Our third experiment compares the actual accuracy of FreeRS, VF, SAS-LC and INS in spread estimation by feeding all of them with the same packet stream that INS can handle. In order for this experiment to work, we have to drop the constraint of (ϵ, β)-RE accuracy and use the original solutions of FreeRS [21] and VF [24], instead of our versions built on Theorems 2-3.

For each solution, we run it over each dataset for 1000 times with different random seeds, calculate the empirical RE for each flow from the 1000 instance values of the error, and check whether (2) holds for each flow whose actual spread is greater than or equal to β. The experimental results are presented in Tables IV-VI for the normal dataset and the skewed dataset, respectively. The flows are divided into a number of bins based on their spreads. The bins cover the spread ranges of $(0, 1], (1, 2], (2, 4], (4, 8], ..., (2^i, 2^{i+1}]$..., respectively. We only present the bins whose ranges are above β and may merge some bins to make the tables concise. For each bin, we show the *pecentages* of its flows that meet (2), i.e., their empirical RE values are bounded by ϵ.

TABLE VIII: Percentage of flows in each bin with their empirical REs bounded by ϵ, over the first 2.8M distinct packets of the normal dataset. Memory allocated is 6.4Mbits, $\beta = 32$ and $\epsilon = 0.1$.

Bin	$[2^5,2^6]$	$(2^6,2^7]$	$(2^7,2^8]$	$(2^8,2^9]$	$(2^9,2^{14}]$	$> 2^{14}$
No. of flows	5656	2225	700	249	298	6
FreeRS	5.7%	65.1%	95.8%	100%	100%	100%
VF	11.4%	36.5%	49.4%	54.6%	60.4%	100%
SAS-LC	35.9%	84.2%	85.5%	87.9%	91.9%	100%
INS	99.8%	100%	100%	100%	100%	100%

Table IV shows the results over the normal dataset when memory allocation is 6.4Mbits, β = 5 and ϵ = 0.1. We use the first 400k distinct packets from the normal dataset, which can be handled by by INS with the given parameter values of β and ϵ. We choose the system parameters for VF, FreeRS and SAS-LC based on their original papers. The results show INS outperforms FreeRS, VF and SAS-LC in estimation accuracy. Specifically, for INS, the empirical REs of all flows in all bins except for $[5, 8]$ are within the bound ϵ. For the bin $[5, 8]$, the empirical REs of 96.4% flows are within the bound, where an empirical RE based on measured data could deviate slightly from the true mean. In contrast, for FreeRS, its percentages of flows that meet (2) are only 10.8% and 48.1% in the bins [5, 8] and [8, 16), respectively, which contain the vast majority of all flows. VF and SAS-LC perform much worse. For VF, its percentages of meeting (2) are zeros for the bins [5, 8], [8, 16), and [16, 32). For SAS-LC, its percentages of meeting (2) for those bins are zero, zero and 2.9%, respectively. One may argue that only the large flows are important, which we do not agree: First, if a user sets $\beta = 5$ as in the experiment, it means that flows of spreads 5 and larger are all important. Second, stealthy attacks may indeed be performed with small or modest spreads [1], [34], and measuring them over time can reveal persistence, changes or other patterns for attack detection. If we increase β to focus on larger flows, we observe similar comparison. Table VIII shows the results of the same experiment except for $\beta = 32$. INS ensures (ϵ, β)-RE accuracy consistently across all bins. FreeRS cannot do that for bins up to (128, 256], VF up to $(2^9, 2^{14}]$, and SAS-LC up to $(2^9, 2^{14}]$.

Table V shows the results over the skewed dataset when memory allocation is 6.4Mbits, β = 5 and ϵ = 0.1. We use the first 750k distinct packets because INS can handle a larger stream size for the skewed dataset, as shown in Table V. INS still ensures (ϵ, β)-RE accuracy consistently across all bins. The performance of FreeRS is much worse. Its percentages of meeting (2) for the bins of [5, 8], (8, 16] and (16, 32] are zero, zero and 24.1%, respectively.

Table VI shows the results under the same parameter setting as above except for $\beta = 32$. Again, INS significantly outperforms FreeRS, VF and SAS-LC in spread estimation accuracy.

VI. Related Work

The most related work is the solutions based on non-duplicate sampling [22] [23], [25], where the most recent work called Virtual Filter (VF) [24] achieves the best memory efficiency and lowest processing overhead. In the context of this paper, the above work is also called uniform non-duplicate sampling. Before the formal definition of non-duplicate sampling, Ting designed a similar method for spread measurement [20]. It maintains a data structure, e.g., a bitmap and an HLL [12], [13], to filter each packet. Only packets of the first appearance have a probability to make changes in the data structure, where the probability p will only decrease as more distinct packets are recorded. In general, this method is also based on non-duplicate sampling if we relax p to be variable. It can be categorized into non-uniform nonduplicate sampling. It is proposed to estimate the spread of a single flow. FreeBS and FreeRS in [21] apply this method to per-flow spread estimation using a large bitmap and a large HLL, respectively. However, the accuracy of flows will be affected by the appearance order. If a flow appears in the tail of the data stream, its sampling probability is small and will be estimated inaccurately.

There is another thread of solutions based on sketches for spread measurement [14], [15], [16], [17], [18], [19], [35], [36]. They use compact data structures for packet recording and produce approximate spread estimates. While they can fit in the limited on-chip memory, they make sacrifices in estimation accuracy by allowing multiple flows to be mapped to the same spread estimator [15], [18], [19], [35] (or a part of the spread estimator [14], [16], [17], [18], [19]).

VII. Conclusion

This paper introduces a new (ϵ, β)-RE accuracy model for spread estimation and a new (ϵ, β)-nonduplicate sampling type, and theoretically establishes their equivalency. We provide theoretical instruction for existing solutions based on non-duplicate sampling to support (ϵ, β)-nonduplicate sampling but we observe they are inefficient in doing so. This paper designs a novel, efficient algorithm based on individualized per-flow sampling with theoretical analysis of the correctness, optimal parameter setting and memory consumption. Our experimental results based on real Internet traffic traces demonstrate that the proposed solution outperforms existing sampling-based solutions significantly in terms of maximum supported packet stream size under a given (ϵ, β)-RE accuracy requirement or in terms of accuracy with the same packet stream size, and demonstrate the superior accuracy performance over existing the state-of-the-art sketch-based solutions to spread estimation.

Appendix. Theorem Proofs

Proof of Theorem 1 We consider an arbitrary flow containing q distinct packets. Let L_q with $q \geq 1$ be the number of sampled packets by (ϵ, β)-nonduplicate sampling if q distinct packets are loaded. Define variable Y_q as

$$\forall a > 1, Y_q = \prod_{j=0}^{L_q} (1 + (a-1)p_{j-1}^{-1}),$$

where p_j is the sampling probability after the jth packet and before $(j+1)$th packet are sampled. This is the finest granularity we can consider— we cannot change the probability each time when we load a new distinct packet because we do not know the packet is a new distinct packet or not. We use p_{-1}, which can be any constant.

Apparently, if $L_q = i$, we have

$$L_{q+1} = i + Bernoulli(p_i).$$

Furthermore, if $L_q = i$, we have

$$\begin{aligned} &\mathrm{E}(Y_{q+1}|Y_0, ..., Y_q) = \mathrm{E}[Y_q|(L_{q+1} = i) \\ &+ Y_q(1 + (a-1)p_i^{-1})|L_{q+1} = i+1)] \\ &= Y_q[1 - p_i + p_i(1 + (a-1)p_i^{-1})] = Y_q a \end{aligned} \tag{8}$$

Therefore, $\{a^{-q}Y_q : q = 0, 1, ...\}$ is a martingale and we have

$$\mathrm{E}(Y_q) = a^q(1 - (a-1)p_{-1}^{-1}).$$

That means

$$a^q(1 - (a-1)p_{-1}^{-1}) = \mathrm{E}(\prod_{j=0}^{L_q}(1 - (a-1)p_{j-1}^{-1})). \tag{9}$$

Before we continue with the above equation, we consider m_k. Define $T_k = m_k - m_{k-1}$ and let $m_0 = 0$. Since each distinct packet is sampled independently with probability of p_{k-1}, T_k complies with the geometric distribution, i.e.,$\Pr(T_k = i) = (1-p_{k-1})^{i-1}p_{k-1}$. According to the property of the geometric distribution, we obtain the expectation and variance of T_k, $k \geq 1$ as

$$\mathrm{E}(T_k) = p_{k-1}^{-1} \tag{10}$$

$$\mathrm{Var}(T_k) = (1 - p_{k-1})p_{k-1}^{-2}. \tag{11}$$

The expectation and variance of m_{L_q} can be calculated as

$$\begin{cases} \mathrm{E}(m_{L_q}|L_q) = \mathrm{E}(\sum_{k=1}^{L_q}(T_k)) = \sum_{k=1}^{L_q} p_{k-1}^{-1} \\ \mathrm{Var}(m_{L_q}|L_q) = \mathrm{Var}(\sum_{k=1}^{L_q} T_k) = \sum_{k=1}^{L_q}(1-p_{k-1})p_{k-1}^{-2} \end{cases} \tag{12}$$

Since (ϵ, β)-nonduplicate sampling requires $RE(m_{L_q}) \leq \epsilon$. Let $\epsilon = \sqrt{C}^{-1}$. Without loss of generality, we set $RE(m_{L_q}) = \epsilon = \sqrt{C}^{-1}$. From (4), we have

$$\mathrm{Var}(m_{L_q}|L_q) = \frac{\mathrm{E}^2(m_{L_q}|L_q)}{C}.$$

Combining with (12), we know

$$\sum_{k=1}^{L_q} \frac{(1-p_{k-1})}{p_{k-1}^2} = \sum_{k=1}^{L_q} (p_{k-1}^{-2} - p_{k-1}^{-1}) = \frac{\mathrm{E}^2(m_{L_q}|L_q)}{C}.$$

By some calculation, we have

$$\sum\nolimits_{k=1}^{L_q} (p_{k-1}^{-2}) = \frac{\mathrm{E}^2(m_{L_q}|L_q)}{C} + \mathrm{E}(m_{L_q}|L_q).$$

This result will be used in the following.

Taking first derivative for (9) at $a = 1_+$, we have $q + p_{-1}^{-1} = \mathrm{E}\sum_{j=0}^{L_q} p_{j-1}^{-1} = \mathrm{E}[\mathrm{E}(m_q|L_q)] + p_{-1}^{-1}$. Therefore,

$$\mathrm{E}[\mathrm{E}(m_{L_q}|L_q)] = q. \tag{13}$$

This means $\mathrm{E}(m_{L_q}|L_q)$ is an unbiased estimate of q.

Taking the second derivative at $a = 1_+$, we have

$$q(q-1) + 2qp_{-1}^{-1} = \mathrm{E}(\sum_{j=0}^{L_q} p_{j-1}^{-1})^2 - \mathrm{E}(\sum_{j=0}^{L_q} p_{j-1}^{-2})$$
$$= \mathrm{E}(\mathrm{E}(m_{L_q}|L_q) + p_{-1}^{-1})^2 - \mathrm{E}(p_{-1}^{-2}$$
$$+ \mathrm{E}(m_{L_q}|L_q) + C^{-1}\mathrm{E}^2(m_{L_q}|L_q)). \quad (14)$$

Adapting (13) to the above equation, we have $\mathrm{E}(\mathrm{E}^2(m_{L_q}|L_q)) = q^2C/(C-1)$. Thus, we have

$$\begin{cases} \mathrm{Var}(\mathrm{E}(m_{L_q}|L_q)) = q^2/(C-1) \\ RE(\mathrm{E}(m_{L_q}|L_q)) = \sqrt{(C-1)^{-1}}. \end{cases} \quad (15)$$

Our above analysis applies for arbitrary sub-stream with q distinct packets. For a special case of flow f with s_f distinct packets and c_f sampled packets, if we use $\mathrm{E}(m_{c_f}|c_f)$ to estimate s_f, The estimate $\hat{s}_f$ satisfies $RE(\hat{s}_f) = \sqrt{(C-1)^{-1}} = \epsilon\sqrt{1/(1-\epsilon^2)}$. The theorem holds.

Proof of Theorem 2 By setting $p_i = p, \forall i \geq 0$ in (12) and adapting them to (4), we can calculate $\mathrm{RE}(m_k)$ as

$$\mathrm{RE}(m_k) = \frac{\sqrt{\mathrm{Var}(m_k)}}{\mathrm{E}(m_k)} = \sqrt{(1-p)}k^{-\frac{1}{2}}.$$

(ϵ, β)-nonduplicate sampling requires that $RE(m_k) \leq \epsilon$ for any flow with spread of $< \beta$. Those flows are expected to have βp^{-1} sampled packets. By adapting $RE(m_k) \leq \epsilon$ and $k = \beta p^{-1}$ to the above equation, we obtain $p \geq \frac{1}{1+\epsilon^2\beta}$. The theorem holds.

Proof of Theorem 3 For any flow f, if $s_f \geq \beta$, $\min\{\frac{1}{1+\epsilon^2 s_f}, \frac{1}{1+\epsilon^2\beta}\} = \frac{1}{1+\epsilon^2 s_f}$. According to Theorem 2, we know that setting $p \geq \frac{1}{1+\epsilon^2 s_f}$ implements (ϵ, s_f)-nonduplicate sampling and therefore by definition we have $RE(m_f^*) \leq \epsilon$. If $s_f < \beta$, it is excluded from the target set of flows whose accuracy should guaranteed in (ϵ, β)-nonduplicate sampling. Therefore, for any flow f, as long as $p_f \geq \min\{\frac{1}{1+\epsilon^2 s_f}, \frac{1}{1+\epsilon^2\beta}\}$, (ϵ, β)-nonduplicate sampling is implemented.

Proof of Theorem 4 We prove by induction. Consider the process producing the $(k+1)$th sampled packets in flow f with $k \geq \bar{k}$. We need to ensure $RE(m_{k+1}) = RE(m_k) = \epsilon$. Let $t_k = \mathrm{E}(m_k)$ and $T_k = m_k - m_{k-1}$. Combining with (4), we have

$$\frac{\mathrm{Var}(m_{k+1})}{t_{k+1}^2} = \frac{\mathrm{Var}(m_k)}{t_k^2} = \epsilon^2 = \frac{\mathrm{Var}(m_k) + \mathrm{Var}(T_k)}{(t_k + \mathrm{E}(T_k))^2}.$$

The above equation can be transformed to $\frac{\epsilon^2 t_k^2 + \mathrm{Var}(T_k)}{(t_k + \mathrm{E}(T_k))^2} = \epsilon^2$.

Adapting (10) and (11) to the above equation, we have

$$\frac{\epsilon^2 t_k^2 + (1-p_f(k))p_f(k)^{-2}}{(t_k + p_f(k)^{-1})^2} = \epsilon^2 \Leftrightarrow p_f(k) = \frac{1-\epsilon^2}{2t_k\epsilon^2 + 1}. \quad (16)$$

Now let's consider the value of t_k. Since

$$t_k = t_{k-1} + \mathrm{E}(T_k) = t_{k-1} + [p_f(k-1)]^{-1},$$

combining with (16), we can obtain t_{k-1} as

$$t_k = \frac{1+\epsilon^2}{1-\epsilon^2}t_{k-1} + \frac{1}{1-\epsilon^2}.$$

This equation is recursive. Since the initial value is $t_{\bar{k}} = (1+\epsilon^2\beta)\bar{k}$, we can derive the value of t_k, $\forall k \geq 1$.$\mathrm{E}(m_k) = t_k$ is $(1+\epsilon^2\beta)k$, if $0 < k \leq \bar{k}$, and is $(\frac{1+\epsilon^2}{1-\epsilon^2})^{k-\bar{k}}t_{\bar{k}} + \frac{1-(\frac{1+\epsilon^2}{1-\epsilon^2})^{k-\bar{k}}}{-2\epsilon^2}$ otherwise. The above probability setting achieves (ϵ, β)-nonduplicate sampling. Specifically, in the boundary case where $s_f = \beta$, it is expected to have $\beta p_\beta \leq \bar{k}$ sampled packets. As a result, it will only experience the sampling probability of p_β, leading to (ϵ, β)-nonduplicate sampling as proven in Theorem 3. For $s_f > \beta$, our parameter setting ensures that $RE(m_{k+1}) = RE(m_k) = \epsilon$ for every $k \geq \bar{k}$. Therefore, (ϵ, β)-nonduplicate sampling always holds.

Proof of Theorem 5 We first prove that the duplicates will be discarded. For any packet $\langle f, e\rangle$, after it passes through Step 1, it is always hashed to $B[H(f,e)]$. It can pass through Step 2 only if $B[H(f,e)] = 0$ when it arrives. In this case, $\langle f, e\rangle$ must be a new packet otherwise its first appearance must (1) be able to pass through Step 1 as the sampling probability in Step 1 p_1 will only decrease — if $h(f,e) < p_1$ now, $h(f,e) < p_1$ must hold at its first appearance; (2) have already set $B[H(f,e)] = 1$, which contradicts to the fact that $B[H(f,e)] = 0$. Therefore, Step 2 can remove duplicates. Next we consider the sampling probability for the packet at its first appearance. It passes through Step 1 with the probability of $p_1 = p_f(c_f)/p_2$. It passes through Step 2 only when $B[H(f,e)] = 0$ and $H(f,e) < \frac{m^2 p_2}{z}$. $B[H(f,e)] = 0$ happens with probability of $\frac{z}{m}$ as there are z bits that are zero and $H(f,e)]$ is a random hash value in the range of $[0, m-1]$. $H(f,e) < \frac{m^2 p_2}{z}$ happens with probability of $\frac{m^2 p_2}{zm} = \frac{mp_2}{z} \leq 1$ as $z \geq mp_2$ before termination. Overall, the total sampling probability is

$$\frac{p_f(c_f)}{p_2} \times \frac{z}{m} \times \frac{mp_2}{z} = p_f(c_f).$$

Proof of Theorem 6 Let n be the expected number of distinct packets to be processed in Alg. 1 with a bitmap with m bits. According to [10], the expected number of distinct packets recorded in the bitmap B is $-m\ln\frac{z}{m}$. The algorithm terminates when $z < mp_2$, at that time the expected number of distinct packets recorded in B in Step 2 is $-m\ln p_2$. Define the average sampling probability P in Step 1 for all flows as $P = -m\ln p_2/n$. As the sampling probability evolves separately for each flow, the value of P varies for different flow distributions. Our parameter setting should maximize the value of n regardless of the flow distribution. The worst case happens when P is maximized, corresponding to the flow distribution that all flows's spread will not exceed β and $p_1 = \frac{p_\beta}{p_2}$ holds all the time. In the case, we have $P = \frac{p_\beta}{p_2} = \frac{-m\ln p_2}{n}$, from which we obtain $n = \frac{-mp_2\ln p_2}{p_\beta}$. Since $p_2 \geq p_\beta$, by some calculation and derivation, the maximum of n can be obtained as follows.

$$n = \begin{cases} \frac{m}{p_\beta e}, & \text{where } p_2 = 1/e; \text{ if } 0 \leq p_\beta < 1/e \\ -m\ln p_\beta, & \text{where } p_2 = p_\beta; \text{ if } \frac{1}{e} \leq p_\beta \leq 1. \end{cases} \quad (17)$$

Therefore, the optimal setting for p_2 is $p_2 = \max\{p_\beta, 1/e\}$.

ACKNOWLEDGMENT

This work is supported by the University of Kentucky start-up fund. Shigang Chen is the corresponding author.

REFERENCES

[1] Y. Gao, Y. Zhao, R. Schweller, S. Venkataraman, Y. Chen, D. Song, and M. Kao, "Detecting Stealthy Spreaders Using Online Outdegree Histograms," *Proc. of IEEE International Workshop on Quality of Service'07*, pp. 145–153, June 2007.

[2] A. Chen, L. Li, and J. Cao, "Estimating Cardinality Distributions in Network Traffic," in *Proceedings of the 2008 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, 2008, pp. 459–460.

[3] A. Yaar, A. Perrig, and D. Song, "Pi: A Path Identification Mechanism to Defend against DDoS Attacks," *Proc. of IEEE Symposium on Security and Privacy*, May 2003.

[4] X. Wang, M. Chen, T. Taleb, A. Ksentini, and V. C. Leung, "Cache in the Air: Exploiting Content Caching and Delivery Techniques for 5G Systems," *IEEE Communications Magazine*, vol. 52, no. 2, pp. 131–139, 2014.

[5] S. K. Fayaz, Y. Tobioka, V. Sekar, and M. Bailey, "Bohatei: Flexible and Elastic DDOS Defense," in *24th {USENIX} Security Symposium ({USENIX} Security 15)*, 2015, pp. 817–832.

[6] W. Jiang, G. Feng, and S. Qin, "Optimal Cooperative Content Caching and Delivery Policy for Heterogeneous Cellular Networks," *IEEE Transactions on Mobile Computing*, vol. 16, no. 5, pp. 1382–1393, 2016.

[7] A. Praseed and P. S. Thilagam, "DDoS Attacks at the Application Layer: Challenges and Research Perspectives for Safeguarding Web Applications," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 661–685, 2018.

[8] L. Tang, Q. Huang, and P. P. Lee, "SpreadSketch: Toward Invertible and Network-Wide Detection of Superspreaders," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 1608–1617.

[9] C. Ma, S. Chen, Y. Zhang, Q. Xiao, and O. O. Odegbile, "Super Spreader Identification using Geometric-min Filter," *IEEE/ACM Transactions on Networking*, vol. 30, no. 1, pp. 299–312, 2021.

[10] K. Whang, B. T. Vander-Zanden, and H. M. Taylor, "A Linear-time Probabilistic Counting Algorithm for Database Applications," *ACM Transactions on Database Systems*, vol. 15, no. 2, pp. 208–229, June 1990.

[11] P. Flajolet and G. N. Martin, "Probabilistic counting algorithms for database applications," *Journal of Computer and System Sciences*, vol. 31, pp. 182–209, September 1985.

[12] P. Flajolet, E. Fusy, O. Gandouet, and F. Meunier, "Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm," *Proc. of AOFA*, pp. 127–146, 2007.

[13] S. Heule, M. Nunkesser, and A. Hall, "HyperLogLog in Practice: Algorithmic Engineering of a State-of-The-Art Cardinality Estimation Algorithm," *Proc. of EDBT*, pp. 683–692, 2013.

[14] M. Yoon, T. Li, S. Chen, and J. Peir, "Fit a Compact Spread Estimator in Small High-Speed Memory," *IEEE/ACM Transactions on Networking*, vol. 19, no. 5, pp. 1253–1264, October 2011.

[15] M. Yu, L. Jose, and R. Miao, "Software Defined Traffic Measurement with OpenSketch," *Proc. of USENIX Symposium on Networked Systems Design and Implementation*, 2013.

[16] Q. Xiao, S. Chen, Y. Zhou, M. Chen, J. Luo, T. Li, and Y. Ling, "Cardinality Estimation for Elephant Flows: A Compact Solution based on Virtual Register Sharing," *IEEE/ACM Transactions on Networking*, 2017.

[17] Z. Zhou and B. Hajek, "Per-flow Cardinality Estimation based on Virtual Loglog sketching," in *2019 53rd Annual Conference on Information Sciences and Systems (CISS)*. IEEE, 2019, pp. 1–6.

[18] 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.

[19] H. Wang, C. Ma, O. O. Odegbile, S. Chen, and J.-K. Peir, "Randomized Error Removal for Online Spread Estimation in Data Streaming," *Proceedings of the VLDB Endowment*, vol. 14, no. 6, 2021.

[20] D. Ting, "Streamed Approximate Counting of Distinct Elements: Beating Optimal Batch Methods," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 442–451.

[21] P. Wang, P. Jia, X. Zhang, J. Tao, X. Guan, and D. Towsley, "Utilizing Dynamic Properties of Sharing Bits and Registers to Estimate User Cardinalities Over Time," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 2019, pp. 1094–1105.

[22] Y.-E. Sun, H. Huang, C. Ma, S. Chen, Y. Du, and Q. Xiao, "Online Spread Estimation with Non-duplicate Sampling," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 2440–2448.

[23] H. Huang, Y.-E. Sun, C. Ma, S. Chen, Y. Du, H. Wang, and Q. Xiao, "Spread Estimation with Non-duplicate Sampling in High-speed Networks," *IEEE/ACM Transactions on Networking*, vol. 29, no. 5, pp. 2073–2086, 2021.

[24] C. Ma, H. Wang, O. O. Odegbile, and S. Chen, "Virtual Filter for Non-duplicate Sampling," in *2021 IEEE 29th International Conference on Network Protocols (ICNP)*. IEEE, 2021, pp. 1–11.

[25] Y. Du, H. Huang, Y.-E. Sun, S. Chen, and G. Gao, "Self-adaptive Sampling for Network Traffic Measurement," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, 2021, pp. 1–10.

[26] Z. Durumeric, M. Bailey, and J. A. Halderman, "An Internet-wide View of Internet-wide Scanning," in *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, 2014, pp. 65–78.

[27] S. Singh, C. Estan, G. Varghese, and S. Savage, "Automated Worm Fingerprinting." in *OSDI*, vol. 4, 2004, pp. 4–4.

[28] S. Sen and J. Wang, "Analyzing Peer-to-peer Traffic Across Large Networks," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*. ACM, 2002, pp. 137–150.

[29] C. J. Willmott and K. Matsuura, "On the Use of Dimensioned Measures of Error to Evaluate the Performance of Spatial Interpolators," *International Journal of Geographical Information Science*, vol. 20, no. 1, pp. 89–102, 2006.

[30] R. J. Hyndman and A. B. Koehler, "Another Look at Measures of Forecast Accuracy," *International journal of forecasting*, vol. 22, no. 4, pp. 679–688, 2006.

[31] R. G. Pontius, O. Thontteh, and H. Chen, "Components of Information for Multiple Resolution Comparison between Maps That Share a Real Variable," *Environmental and ecological statistics*, vol. 15, no. 2, pp. 111–142, 2008.

[32] Wikipedia, "Section 1.2.1 of Mean Squared Error: Proof of Variance and Bias Relationship," https://en.wikipedia.org/wiki/Mean_squared_error, 2022.

[33] UCSD., "Caida ucsd anonymized 2015 internet traces on jan. 17," https://www.caida.org/data/passive/passive_2015_dataset.xml, 2015.

[34] Q. Xiao, Y. Qiao, M. Zhen, and S. Chen, "Estimating the Persistent Spreads in High-speed Networks," in *2014 IEEE 22nd International Conference on Network Protocols*. IEEE, 2014, pp. 131–142.

[35] C. Ma, O. O. Odegbile, D. Melissourgos, H. Wang, and S. Chen, "From countmin to super kjoin sketches for flow spread estimation," *IEEE Transactions on Network Science and Engineering*, 2023.

[36] X. Song, J. Zheng, H. Qian, S. Zhao, H. Zhang, X. Pan, and G. Chen, "Couper: Memory-efficient cardinality estimation under unbalanced distribution," in *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 2023, pp. 2753–2765.