IEEE INFOCOM 2021 - IEEE Conference on Computer Communications | 978-1-6654-0325-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/INFOCOM42981.2021.9488425

Self-Adaptive Sampling for Network Traffic Measurement

Yang Du[†], He Huang[†], Yu-E Sun[‡], Shigang Chen[§], Guoju Gao[†]
[†]School of Computer Science and Technology, Soochow University, Suzhou, China
[‡]School of Rail Transportation, Soochow University, Suzhou, China
[§]Department of Computer and Information of Science and Engineering, University of Florida, US
E-mail: huangh@suda.edu.cn
*He Huang is the corresponding author.

***Abstract*—Per-flow traffic measurement in the high-speed network plays an important role in many practical applications. Due to the limited on-chip memory and the mismatch between off-chip memory speed and line rate, sampling-based methods select and forward a part of flow traffic to off-chip memory, complementing sketch-based solutions in estimation accuracy and online query support. However, most current work uses the same sampling probability for all flows, overlooking that the sampling rates different flows require to meet the same accuracy constraint are different. It leads to a waste in storage and communication resources. In this paper, we present self-adaptive sampling, a framework to sample each flow with a probability adapted to flow size/spread. Then we propose two algorithms, SAS-LC and SAS-LOG, which are geared towards per-flow spread estimation and per-flow size estimation by using different compression functions. Experimental results based on real Internet traces show that, when compared to NDS in per-flow spread estimation, SAS-LC can save around 10% on-chip space and reduce up to 40% communication cost for large flows. Moreover, SAS-LOG can save 40% on-chip space and reduce up to 96% communication cost for large flows than NDS in per-flow size estimation.**

Index Terms*—*Traffic measurement, self-adaptive sampling, size estimation, spread estimation.

I. Introduction

Per-flow traffic measurement over network data provides indispensable information for applications like resource allocation, anomaly detection, and access profiling [1]–[16]. There are primarily two types of measurements. One is per-flow size measurement, which counts the number of elements in a flow. The other is per-flow spread measurement that measures the number of *distinct* elements. The definitions of flow and element can be flexibly configured to meet the measurement requirements of different applications. For instance, when detecting scanners, we may define a per-source flow as the packets sent from the same source address, and measure the flow spread by counting the number of distinct destinations (elements) that this source has contacted [13], [14], [17], [18]. For another example, we may treat each packet as an element and measure the flow size as the number of packets, which helps find heavy-hitters [19]–[22].

Due to space and processing speed constraints, it is challenging to implement per-flow traffic measurement at high-speed network links (*e.g.*, 40Gbps). For example, the one-hour Internet trace downloaded from CAIDA [23] contains millions of per-source flows. It is almost impossible to maintain a separated counter for each flow in the limited on-chip memory like SRAM (usually less than 8.25MB) [24]. To solve this problem, sketch-based methods use compact data structures, *i.e.*, sketches, to store flow traffic so that they fit in limited on-chip memory [25]–[28]. However, recent work [11] has pointed out that sketch-based methods only support offline queries and show low accuracy for small flows. Unlike sketch-based solutions, sampling-based methods maintain a separated counter for each flow in the off-chip memory, improving estimation accuracy for small flows and supporting online traffic queries [1], [11], [29]–[33]. Notice that, due to the mismatch between line speed and off-chip memory speed, they require an on-chip sampling module to sample the flow traffic and forward the sampled data to off-chip memory.

A major problem of sampling-based methods is that they often use a same sampling rate for all flows, regardless of the flow size/spread. As pointed out in [24], [34], [35], using a same sampling rate will result in either low accuracy for small flows or massive communication overhead. We must stress that an accurate estimation for small flows is indispensable. It can provide valuable information for detecting stealthy scanner or stealthy DDoS attackers that operate in low-profile manners [4], [36].

We aim to complement prior work by sampling each flow with a probability adapted to its size/spread. This idea is motivated by our observation that, when setting the same accuracy constraint (*e.g.*, the relative bias is below a certain threshold) for all flows, the sampling rates that different flows require to satisfy the constraint are correlated to their sizes/spreads. For instance, if we expect the mean relative bias of estimated flow size to be less than 0.1, the required sampling rates of two flows (size are 100 and 1000) are 0.39 and 0.06, respectively. Clearly, sampling flows with adaptive probabilities can save on-chip memory and reduce communication costs.

It is, however, tricky to assign adaptive sampling probabilities to different flows. Ideally, if we know each flow's size or spread, then for each flow, we can select a minimal sampling probability and sample the flow elements with selected probability. But in practice, we do not know the actual flow size/spread when performing a measurement. Notice that some recent work deals with this problem by predicting per-flow size/spread and sampling different flows with uneven probabilities [37], [38]. However, due to the dynamicity and uncertainty of network traffic, prediction per-flow traffic is itself a challenging problem. Hence there are no performance guarantees for the measurement results of these methods.

In this paper, we present *self-adaptive sampling*, a novel framework for per-flow traffic measurement, which samples each flow with an adaptive probability when holding no assumption for its actual size/spread. This framework adopts an on-chip/off-chip design where a self-adaptive sampling module is placed on the network processor chip to catch up with the line rate, and an off-chip recording module is designed to store flow traffic. The key to this design is *flow compression*. It adaptively compresses the flows by mapping flow elements to virtual elements. Then we employ *non-duplicate sampling* [11] to remove the duplicated virtual elements and sample each distinct virtual element with a pre-defined probability. This allows us to customize the self-adaptive sampling using different compression functions and sampling rates, meeting the requirements of different applications. We propose two algorithms, SAS-LC and SAS-LOG, which are geared towards per-flow spread estimation and per-flow size estimation by using different compression functions. We also perform extensive experiments on real Internet traffic traces downloaded from CAIDA [23]. The experimental results show that our design is efficient and highly configurable to meet different applications' interests.

II. Preliminary

A. Problem statement

We consider the packet stream $\mathcal{P} = \{\mathbb{P}_1, \mathbb{P}_2, \mathbb{P}_3, \cdots\}$ during a measurement epoch, where each packet $\mathbb{P} \in \mathcal{P}$ carries a flow label f (*e.g.*, source/destination address) and an element label e (*e.g.*, packet or destination/source address). The definitions of flow and element can be flexibly configured according to the measurement requirements. We model the packet stream as a set of flows $\mathcal{F} = \{f_1, f_2, f_3, \cdots\}$, where each flow f_i consists of all the packets carrying flow label f_i.

The objective of per-flow traffic measurement is to measure each flow in terms of flow size (number of elements) or flow spread (number of *distinct* elements). Given a set of flows $\mathcal{F}$ and a packet stream $\mathcal{P}$, let $n_{f_1}, n_{f_2}, n_{f_3}, \cdots$ be the actual flow sizes/spreads of flows $f \in \mathcal{F}$. The outputs of per-flow traffic measurement are the estimations for flow sizes/spreads, which are $\widehat{n_{f_1}}, \widehat{n_{f_2}}, \widehat{n_{f_3}}, \cdots$.

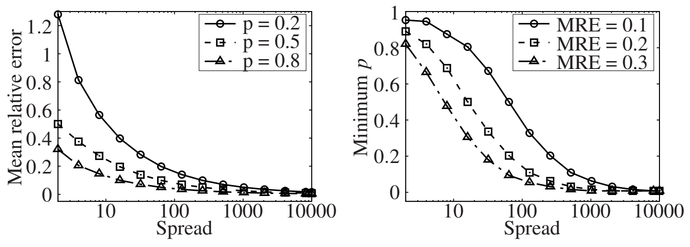
B. Prior art and limitations

There are primarily two types of solutions for per-flow traffic measurement: *sketch-based* and *sampling-based*.

Sketch-based solutions [25]–[28] use compact data structures (*i.e.*, sketches like CM, Bitmap, HLL) to store flow traffic and reduce memory usage, which can fit in limited on-chip memory. However, their model choice of placing the sketches entirely in on-chip memory results in two limitations. First, they require scanning hundreds or thousands of bits/registers when estimating the flow size/spread, making it only support offline queries. Second, they have to make sacrifices in estimation accuracy to achieve high space efficiency, especially when measuring small flows.

The second kind of solution is based on sampling, which has been widely adopted for per-flow size/spread measurement [1], [11], [29]–[33]. Unlike sketch-based solutions, sampling-based solutions use off-chip memory to maintain a separated counter for each flow, preventing the noises introduced by bit/register sharing and supporting online queries. Due to the gap between line speed and off-chip memory speed, an on-chip sampling module is required to process packet stream at line speed, which selects and forwards a part of flow traffic to off-chip memory.

Most existing sampling-based solutions sample all flow elements with the same probability, regardless of flow size/spread. Since the sampling rates of different flows may deviate from the preset probability, a pioneer work named SketchFlow [16] integrates sketches with sampling to provide the same sampling rate across all flows. In addition, as pointed out in [24], [34], [35], using a same sampling rate will result in either low accuracy for small flows or massive communication overhead. Take the spread estimator in [11] as an example. As shown in Fig. 1(a), when setting the same sampling probability for all (distinct) elements, *e.g.*, 0.2, 0.5, or 0.8, large flows' mean relative errors (MRE) are significantly lower than small flows. Fig. 1(b) shows the required sampling probabilities for flows to bound the same MRE (*e.g.*, 0.1, 0.2, and 0.3), which decreases significantly as flow spread grows.



(a) MRE w.r.t. p (b) minimum p w.r.t. MRE

Fig. 1: Relationship between mean relative error (MRE) and sampling rate p.

This motivates us to explore self-adaptive sampling, which samples each flow with a probability adapted to flow size/spread. Notice that, some recent researches [37], [38] sample different flows with adaptive probabilities based on per-flow traffic prediction. However, predicting per-flow traffic is challenging and lacks performance guarantees. In this paper, we choose to implement self-adaptive sampling without predicting per-flow traffic.

C. Our goal

Our goal is to design a self-adaptive sampling framework for network traffic measurement, which works with small on-chip memory and small communication overhead, providing accurate estimations for per-flow sizes/spreads. Our design is desired to have the following properties.

(1) Self-adaptive sampling. Due to the dynamicity and the uncertainty of network traffic, the actual flow sizes/spreads are unknown until the measurement epoch ends. We want our design to adaptively configure a sampling probability for each flow when holding no assumption for the flow size/spread.

(2) Flexibility in measurement. We want our design can flexibly configure the self-adaptive sampling module to meet different measurement needs, *e.g.*, measurement type, measurement range, and measurement performance.

(3) Relative error bounds. We want to provide a probabilistic guarantee for the relative errors of flows with a form similar to [4], [11]: Given a positive integer l, a relative error bound δ, and a probability ϵ $(0 < \epsilon < 1)$, the relative error of a flow, whose size or spread is larger than l, is bounded by δ with probability $1 - \epsilon$.

III. Design of The Self-Adaptive Sampling

A. Main idea

The key idea of *self-adaptive sampling* is simple: *sample each flow's elements with a probability adapted to its size/spread, saving resources while guaranteeing estimation accuracy*. However, it is tricky to determine appropriate sampling probability for each flow during measurement, since the flow size/spread is unknown a priori.

To meet this challenge, we disassemble the task of self-adaptive sampling into two parts: *flow compression* and *non-duplicate sampling*. Flow compression solves the problem that all flows share the same sampling rates by adaptively compressing flows (*i.e.*, reducing flow sizes/spreads), which is achieved by mapping flow elements to virtual elements. Since virtual elements may contain duplicates, we employ non-duplicate sampling [11] to sample each distinct virtual element with the same, pre-defined probability. With this design, we can measure per-flow traffic by counting the number of sampled distinct virtual elements for each flow.

The overall sampling probability of an arbitrary flow will be the ratio between the number of sampled distinct virtual elements and actual flow size/spread. Thus, we can select appropriate compression functions and sampling rates, tuning each flow's sampling probability to be adapted to its flow size/spread, *i.e.*, achieving self-adaptive sampling.

B. Architecture

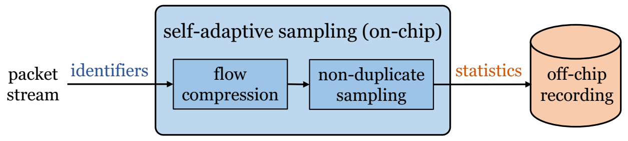


Fig. 2: The system model of self-adaptive sampling

A descriptive architecture of self-adaptive sampling is presented in Fig. 2. We adopt an on-chip/off-chip model in [11], which contains both on-chip part and off-chip part. We place a self-adaptive sampling module on the network processor chip, composed of two components, *flow compression* and *non-duplicate sampling*. The off-chip part is a recording module that maintains a separate counter for each flow.

The benefit of this on-chip/off-chip design is two-fold: First, by using high-speed on-chip memory, we can process packet stream at line rate. Second, we can maintain a separate counter for each flow with large off-chip memory, reducing the noises introduced by sharing bits/registers.

C. Basic Operations

Our design supports two operations to meet the requirement of per-flow traffic measurement. One is *Recording*, which processes packet stream and updates flow traffic statistics at line speed. The other operation is *Estimation*, which answers the online query for an arbitrary flow's size or spread.

1) Recording: For an incoming packet that carries flow label f and element label e, the *Recording* operation is performed as follows: First, *flow compression* transforms element e to a virtual element e'. Then, *non-duplicate sampling* checks if the virtual element has been seen before. If e' is a new virtual element, it samples e' with a pre-defined probability. When a virtual element is sampled, *off-chip recording* will be triggered to update the flow statistics, *e.g.*, sending flow label f to off-chip module and increasing flow f's counter value by one.

2) Estimation: When the measurement epoch ends, we can estimate the sizes or spreads for all flows. Given an arbitrary flow label, we will first lookup the record entry in *off-chip recording*. If no record matches, we regard this flow as an empty flow. Otherwise, we can estimate the actual size/spread based on the compression function and sampling probability.

D. Flow compression

Flow compression aims to adaptively compress flows, which reduces the flow sizes/spreads, solving the problem that all flows share a same sampling rate. We want to stress that, flow compression works for both per-flow size estimation and per-flow spread estimation, but in different manners. Particularly, when performing per-flow size estimation, each flow element (*e.g.*, packet) is treated as a distinct one, which can be considered as a particular case of spread estimation.

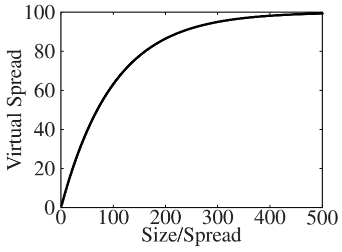
Let $\mathcal{FC}$ denote the compression function. When compressing a flow f with size/spread n_f, it maps each element in flow f to a virtual element, where the virtual elements that $\mathcal{FC}$ outputs form a compressed flow f'. Notice that, different elements may be mapped to a same virtual element. Thus, $\mathcal{FC}$ transforms the n_f (distinct) elements $\{e_1, e_2, \cdots, e_{n_f}\}$ of flow f to a set of $n_{f'}$ distinct virtual elements, which is denoted by $\{e'_1, e'_2, \cdots, e'_{n_{f'}}\}$. Formally:

$$\mathcal{FC}(\{e_1, e_2, \cdots, e_{n_f}\}) \rightarrow \{e'_1, e'_2, \cdots, e'_{n_{f'}}\}. \qquad (1)$$

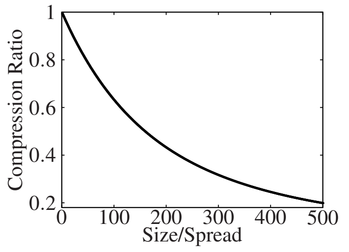
We use virtual spread $n_{f'}$ to represent the number of distinct virtual elements in the compressed flow f'. Clearly, there exists a one-to-one correspondence between virtual spread and flow size/spread, with which we can estimate the actual flow size/spread based on the virtual spread. Suppose that $\mathcal{FC}^{-1}$

is the inverse function of $\mathcal{FC}$. In contrast to $\mathcal{FC}$, $\mathcal{FC}^{-1}$ takes virtual spread $n_{f'}$ as input and returns an estimation for the original flow size/spread n_f. Formally:

$$\widehat{n_f} = \mathcal{FC}^{-1}(n_{f'}). \tag{2}$$



(a) Virtual spread w.r.t. size/spread



(b) Compression ratio w.r.t. size/spread

Fig. 3: Example of a flow compression function.

Consider a compression function $\mathcal{FC}_{100}(\cdot)$ which randomly maps (distinct) elements to 100 virtual elements, $\{1, 2, \cdots, 100\}$, with a same probability, *i.e.*, $\frac{1}{100}$. Fig. 3(a) shows the relationship between flow size/spread and the virtual spread when applying $\mathcal{FC}_{100}$. We observe that the expected virtual spread increases when flow size grows, and there exists a one-to-one correspondence, which means we can estimate the actual flow size/spread based on the virtual spread. In Fig. 3(b), we show the curve of the compression ratio, which refers to the ratio between virtual spread and actual size/spread. We find out that the compression ratio decreases as flow size/spread grows. This feature can help us adaptively assign lower sampling rates to larger flows, achieving our design goal.

Notice that the number of virtual elements is the same as the number of flow elements. We cannot directly download the virtual elements to off-chip memory since it is inefficient and can waste storage and communication resources. Instead, we employ non-duplicate sampling to filter the duplicates and select a subset of distinct virtual elements for off-chip recording.

E. Non-duplicate sampling

We implement non-duplicate sampling based on [11], which uses a bit array B of M bits to sample each distinct virtual element with a pre-defined probability p at its first appearance. The inputs of this module are the virtual elements that flow compression generates. The outputs are the sampled distinct virtual elements along with flow labels.

Given a virtual element, non-duplicate sampling will first check if the element has been seen before. In detail, it initializes all the bits in B as zeros when measurement starts, pseudo-randomly maps each virtual element (with flow label) to a bit h in the bit array B, and regards a virtual element as a new one only if $B[h] = 0$. Notice that, whenever seeing a new virtual element, it will set the corresponding bit to 1, which ensures all subsequent appearances of this virtual element will be identified as duplicates. Due to hash collisions, multiple virtual elements may be hashed onto the same bit, which may incur false positives, *i.e.*, misidentifying a new virtual element as a duplicate. Therefore, when a virtual element is mapped to a bit of 0, non-duplicate sampling samples it with a probability $p' = \frac{p}{V_0}$, where V_0 is the fraction of zeros in B. This ensures that, for each new virtual element, its probability of being hashed onto a bit of zero and selected will be $V_0 \times p' = p$, *i.e.*, the pre-defined sampling probability.

IV. Algorithm Design

This section presents two algorithms for self-adaptive sampling, which are: self-adaptive sampling with linear compression (SAS-LC) and self-adaptive sampling with logarithmic compression (SAS-LOG).

A. Self-adaptive sampling with linear compression

We implement SAS-LC based on a linear compression function $\mathcal{FC}_s$, which maps flow elements to s virtual elements with the same probability, *i.e.*, $\frac{1}{s}$.

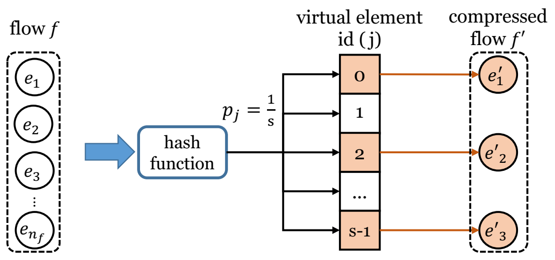


Fig. 4: An example of linear compression

1) Flow Compression: As depicted in Fig. 4, when compressing a flow with size/spread n_f, $\mathcal{FC}_s$ first operates a hash $H(f \oplus e)$ on each element e and maps e to a virtual element e' by $e' = H(f \oplus e) \mod s$, where H is a hash function and $\oplus$ is the XOR operation. In this case, p_j, the probability of mapping an arbitrary element e to a virtual element $e' = j, j \in [0, s-1]$ is always $\frac{1}{s}$. Formally, $\mathcal{FC}_s$ is performed as follows:

$$\mathcal{FC}_s(f, e) = H(f \oplus e) \mod s. \tag{3}$$

To demonstrate the execution of $\mathcal{FC}_s$, we use Fig. 4 as an example, where a flow f contains n_f (distinct) elements, *i.e.*, flow size or flow spread is n_f. By operating $\mathcal{FC}_s$ on each element, we obtain three different outputs $\{0, 2, s-1\}$, denoting the IDs of virtual elements. In other words, $\mathcal{FC}_s$ compresses a flow f with size/spread n_f to a compressed flow f' with virtual spread 3, where f' is composed of three distinct elements $\{e'_1, e'_2, e'_3\}$ with IDs $\{0, 2, s-1\}$.

Based on Linear Counting [39], we know the expected virtual spread is correlated to the flow size/spread, which can be obtained by $n_{f'} = s - s \cdot e^{-\frac{n_f}{s}}$. Therefore, we can estimate the flow size/spread n_f by:

$$\mathcal{FC}_s^{-1}(n_{f'}) = -s \ln(1 - \frac{n_{f'}}{s}). \tag{4}$$

2) Data Structure: The on-chip data structure contains a system parameter s controlling the compression function $\mathcal{FC}_s$ and a bit array B of M bits to serve for non-duplicate sampling. Besides, we maintain a counter c denoting the number of 1 in B. In the off-chip memory, we maintain a separated counter c_f for each flow f, recording the number of virtual elements this flow has been sampled. The bit array B, on-chip counter c, and off-chip counters c_f are all initialized to zeros at the beginning of measurement epoch.

3) Recording: The recording operation is performed on each flow element. Given an element e of flow f, we first perform $\mathcal{FC}_s$ to obtain a virtual element $e' = \mathcal{FC}_s(f, e)$. Based on [11], we assign virtual element e' with flow label f to a bit h in bit array B, which is computed by $h = H'(f \oplus e') \mod M$. Here H' is an independent hash function.

Given the status of $B[h]$, there are two cases to consider: One is $B[h] = 1$, in which case we regard this element as a duplicate and take no further action; the other is $B[h] = 0$, which means $< f, e' >$ is a new virtual element that has not been seen before. As discussed in Section III-E, when $B[h] = 0$, we will sample this element with a probability p', ensuring the overall sampling probability for a new virtual element is p. The value of p' can be computed as follow,

$$p' = p \cdot \frac{M}{M - c}, \tag{5}$$

where $\frac{M}{M-c}$ is the inverse of V_0, the fraction of zeros in the bit array B.

When an element is selected, we will trigger the off-chip recording and increase flow f's separated counter c_f by 1; when an element is not selected, we will not download the flow label. No matter whether this virtual element is selected, we will set $B[h]$ to 1 and increase the online counter c by 1, which ensures all duplicates of this virtual element will be ignored.

4) Estimation: When querying the size or spread of flow f, we first hash f to find this table entry c_f. If none table entry matches, we regard this flow as an empty flow and return an estimated size/spread 0. If there exists a table entry c_f, according to the property of non-duplicate sampling, we can estimate the virtual spread $n_{f'}$ by dividing c_f with sampling probability p, *i.e.*, $n_{f'} = \frac{c_f}{p}$. Then, according to Equation 4, the size/spread of flow f can be estimated as:

$$\widehat{n_f} = -s \ln(1 - \frac{c_f}{sp}). \tag{6}$$

The problem of SAS-LC is that it has to set a large s to provide accurate estimations for large flows. But setting a large s will result in a high compression ratio for small flows (*e.g.*, close to 1), downgrading the efficiency of SAS-LC. Thus, it is more suitable for the scenarios where flow sizes/spreads are small (*e.g.*, scan detection) while not suitable for heavy-hitter detection since the largest flow may contain millions of packets (elements).

B. Self-adaptive sampling with logarithmic compression

Given SAS-LC's limitation, we present SAS-LOG, which utilizes a logarithmic compression function $\mathcal{FC}_{d,s}$ to provide a broad estimation range with high efficiency.

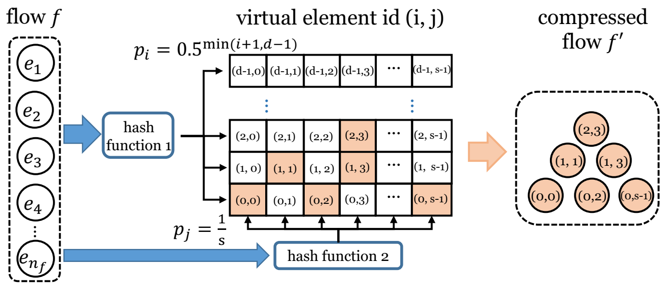

flow f
virtual element id (i, j)
compressed flow f'
$p_i = 0.5^{\min(i+1,d-1)}$
e_1
e_2
e_3
e_4
e_{n_f}
hash function 1
hash function 2
$p_j = \frac{1}{s}$
(d-1,0) (d-1,1) (d-1,2) (d-1,3) ⋯ (d-1, s-1)
(2,0) (2,1) (2,2) (2,3) ⋯ (2, s-1)
(1, 0) (1, 1) (1, 2) (1, 3) ⋯ (1, s-1)
(0,0) (0,1) (0,2) (0,3) ⋯ (0, s-1)
(2,3) (1, 1) (1, 3) (0,0) (0,2) (0,s-1)

Fig. 5: An example of logarithmic compression

1) Flow Compression: We design a compression function $\mathcal{FC}_{d,s}$ parameterized by two integers d and s. As shown in Fig. 5, there are $d \times s$ possible virtual elements, whose IDs are abstracted as a table with d rows and s columns. By using two hash functions, $\mathcal{FC}_{d,s}$ maps each flow element e to a virtual element e' with id (i, j), representing the j-th entry at the i-th row in the ID table. The first hash H is to determine the value of i, *i.e.*, row index, and the second hash H' is to determine the value of j, which is the column index.

Given an element e of flow f, we first determine the row index for its corresponding virtual element. We perform a hash $z = H(f \oplus e) \mod 2^d$ and transform z to a d bit binary form, $< z_0 z_1 z_2 \cdots z_{d-1} >_2$. The row index i is calculated by $i = \rho(z)$. Here $\rho(z)$ returns the order of leftmost 1 in the binary form of z, which starts from 0 and returns $d-1$ when all bits are zeros. By this means, we map each element to the i-th row with a probability $p_i = 0.5^{\min(i+1,d-1)}$ so that $\sum_{i=0}^{d-1} p_i = 1$. For instance, the leftmost 1 in 0111_2 and 0010_2 are the second bit and the third bit, indicating $\rho(0111_2) = 1$ and $\rho(0010_2) = 2$. Then we compute the column index j for by using $j = H'(f \oplus e) \mod s$. Combine above results, we can transform an element e with flow label f to a virtual element (i, j) with a probability of $p_{i,j}$:

$$\mathcal{FC}_{d,s}(f, e) = (i, j), \quad p_{i,j} = 0.5^{\min(i+1,d-1)} \times \frac{1}{s}. \tag{7}$$

Notice that, when compressing a flow, the number of distinct virtual elements at different rows are different. As shown in Fig. 5, $\mathcal{FC}_{d,s}$ maps n_f distinct elements to 6 virtual elements, where the numbers of distinct elements at the first three rows are $\{3, 2, 1\}$. We use $\{n_{f',0}, n_{f',1}, \cdots, n_{f',d-1}\}$ to represent the numbers of distinct virtual elements at d rows, whose sum $\sum_{i=0}^{d-1} n_{f',i}$ is the virtual spread $n_{f'}$. Similar to Equation 4, we can estimate $n_{f,i}$, the number of (distinct) elements that have been mapped to the i-th row by:

$$n_{f,i} = -s \ln(1 - \frac{n_{f',i}}{s}). \tag{8}$$

Since the probabilities of mapping (distinct) elements onto different rows are different, a large flow may fulfill the bottom row but can still obtain an accurate estimation based on higher rows. Thus, for a flow f with virtual spread $n_{f'}$, we will select a starting row $\gamma(n_{f'})$ and only use it and the higher rows to estimate the flow size/spread. The selection of the starting row will be illustrated in Section V-B.

Recall that, the probability of mapping an element to the i-th row is $p_i = 0.5^{\min(i+1,d-1)}$. The number of elements assigned to rows starting from $\gamma(n_{f'})$ will be $\sum_{i=\gamma(n_{f'})}^{d-1} n_{f,i}$, which is $0.5^{\gamma(n_{f'})}$ of the flow size/spread. Therefore, the flow size/spread n_f can be estimated by:

$$\mathcal{FC}_{d,s}^{-1}(n_{f'}) = -s \cdot 2^{\gamma(n_{f'})} \sum_{i=\gamma(n_{f'})}^{d-1} \ln(1 - \frac{n_{f',i}}{s}). \tag{9}$$

2) Data Structure: The on-chip data structure contains two system parameters d, s controlling the compression function $\mathcal{FC}_{d,s}$, a bit array B of M bits, and a counter c. In the off-chip memory, we maintain a separated d-dimensional counter array $c_f = \{c_{f,i}\}_{i=0}^{d-1}$ for each flow f, where $c_{f,i}$ denotes the number of sampled virtual elements with row index i. The bit array B, on-chip counter c, and off-chip counter arrays c_f are all initialized to zeros at the beginning of each measurement epoch.

3) Recording: Similar to SAS-LC, when given an element e of flow f, we first perform $\mathcal{FC}_{d,s}$ on e and obtain a virtual element $e' = (i, j)$. Then assign it a bit $h = H''(H''(f \oplus i) \oplus j) \mod M$ in bit array B, where H'' is another independent hash function. Only when $B[h]$ is 0, we set this bit to 1 and sample virtual element e' with a probability $p' = p \cdot \frac{M}{M-c}$. At last, when a virtual element $e' = (i, j)$ is selected, we will download the flow label f and row index i to off-chip memory, then increase counter $c_{f,i}$ by one.

4) Estimation: When querying the size or spread for a flow f, we first hash f to find its table entry and return its current counter value $c_f = \{c_{f,i}\}_{i=0}^{d-1}$. Recall that we use $\gamma(c_f)$ to denote the starting row. The number of (distinct) elements hashed onto it and higher rows is expected to be the $\frac{1}{2^{\gamma(c_f)}}$ of the total size/spread. Therefore, we can estimate the size/spread for flow f by:

$$\widehat{n_f} = -s \cdot 2^{\gamma(c_f)} \sum_{j=\gamma(c_f)}^{d-1} \ln(1 - \frac{c_{f,j}}{sp}). \tag{10}$$

V. Optimal system parameters

In the following, we present the parameter selection for SAS-LC and SAS-LOG, respectively. Our goal is to select optimal parameters by minimizing the on-chip space requirement M when providing following performance guarantee: given a positive integer l, a relative error bound δ, and a probability value ϵ ($0 < \epsilon < 1$), for a flow with a size or spread larger than l, its relative error is bounded by δ with probability $1-\epsilon$. Notice that, when selecting parameters, we assume the largest flow size/spread is h.

A. System parameters for SAS-LC

When bounding the relative errors for SAS-LC, there are three system parameters M, s, and p to determine. M is the size of the bit array, s is the parameter of compression function, and p is the sampling rate of non-duplicate sampling.

Consider an arbitrary flow f whose size/spread is n_f. We want to bound the relative errors by δ with a probability larger than $1-\epsilon$, *i.e.*, the estimated size/spread $\widehat{n_f}$ satisfies:

$$n_f(1-\delta) \le \widehat{n_f} \le n_f(1+\delta). \tag{11}$$

As discussed in Section III.A, the overall sampling probability of flow f, denoted as p_f, is the product of compression ratio and the sampling rate of non-duplicate sampling, which is:

$$p_f = (1 - (1 - \frac{1}{s})^{n_f}) \frac{sp}{n_f}. \tag{12}$$

Let c_f be the counter value of f's table entry. It is the number of successes in n_f Bernoulli trials when the probability of success is p_f, *i.e.*, c_f follows a Binomial distribution parameterized by n_f and p_f: $c_f \sim \mathrm{B}(n_f, p_f)$. Let $Pr\{c_f = k\}$ denote the probability of $c_f = k$ for $k = 0, 1, \cdots, n_f$, it can be computed by:

$$Pr\{c_f = k\} = C_{n_f}^k p_f^k (1 - p_f)^{n_f - k}. \tag{13}$$

According to Equation 6, we can estimate the flow size/spread based on counter value c_f. When bounding the mean relative error by δ, the range of possible c_f should be:

$$(1 - e^{-\frac{n_f(1-\delta)}{s}})sp \le c_f \le (1 - e^{-\frac{n_f(1+\delta)}{s}})sp. \tag{14}$$

Let $p_\delta(s, p, n)$ denote the probability when the mean relative error of a flow, whose size/spread is n, is less than δ. It is the sum of probabilities $Pr\{c_f = k\}$ when k is within the feasible range as in Equation 14:

$$p_\delta(s, p, n) = \sum_{j=\lceil (1-e^{-\frac{n(1-\delta)}{s}})sp \rceil}^{\lfloor (1-e^{-\frac{n(1+\delta)}{s}})sp \rfloor} C_n^j p_f^j (1 - p_f)^{n-j}. \tag{15}$$

Notice that when the values of s, p, and δ are fixed, $p_\delta(s, p, n)$ becomes a function of n. It is a curve as illustrated in Figure 6, where non-smooth appearance is due to $\lceil \cdot \rceil$ and $\lfloor \cdot \rfloor$ operations in Equation. 15. Approximately, we can say for all $n \in [l, h]$, $p_\delta(s, p, n)$ is always larger than $\min\{p_\delta(s, p, l), p_\delta(s, p, h)\}$.

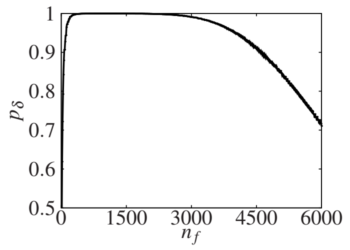


Fig. 6: Curve of $p_\delta(\cdot)$ when $s = 2000$, $p = 0.5$, $\delta = 0.1$

We formalize the parameter selection problem as below. N' refers to the sum of virtual spreads when applying $\mathcal{FC}_s$, which can be obtained from historical data. The first constraint ensures that for each virtual element, its sampling probability p' computed by Equation 5 will be within $(0, 1]$. The second and third constraints bound the mean relative errors for flows whose size/spread are within $[l, h]$ by δ with a probability larger than $1 - \epsilon$.

$$\min M \quad \text{s.t.} \begin{cases} M \geq -\frac{N'}{\ln p} \\ p_\delta(s, p, l) \geq 1 - \epsilon \\ p_\delta(s, p, h) \geq 1 - \epsilon \end{cases} \tag{16}$$

B. System parameters for SAS-LOG

For SAS-LOG, there are four system parameters M, d, s, and p to determine, where M is the size of the bit array, (d, s) are the parameters of compression function, and p is the sampling rate of non-duplicate sampling.

When estimating per-flow size/spread, we should first filter out the rows with inappropriate resolutions. Consider an arbitrary flow f whose spread is n_f. Let $c_f = \{c_{f,0}, c_{f,1}, \cdots, c_{f,d-1}\}$ be the counter values of flow f. Based on Equation 10, we only use the rows starting from $\gamma(c_f)$, *i.e.*, $\gamma(c_f), \gamma(c_f) + 1, \cdots d - 1$, to estimate the size/spread of f.

Let r_f be the number of (distinct) elements that have been hashed onto the rows starting from $\gamma(c_f)$. As proved in [40], when $\gamma(c_f) \leq d - 3$, we can multiply the estimation of r_f with $2^{\gamma(c_f)}$ for an unbiased estimation of n_f, where the estimation error is mainly contributed by $\widehat{r_f}$. Let $p_\delta(d, s, p, n = r_f)$ denote the probability that $\widehat{n_f}$ is distributed in $[n_f(1 - \delta), n_f(1 + \delta)]$. We can compute its value by enumerating the combinations of $c_{f,\gamma(c_f)}, c_{f,\gamma(c_f)+1}, \cdots$.

The curve of $p_\delta(d, s, p, n)$ when fixing d, s, p is similar to Fig. 6, which is approximately a convex function of n. When setting appropriate d, s, and p, we can find the smallest integer l' and the largest integer h' for r_f that makes $p_\delta(d, s, p, r_f)$ larger than $1-\epsilon$. Therefore, for a flow whose size/spread is n_f, if we can find a minimum starting row index $\gamma(c_f) \in [0, d-3]$ that makes $r_f = \frac{n_f}{2^{\gamma(c_f)}}$ be within $[l', h']$, we can say its mean relative error is bounded by δ with a probability of $1 - \epsilon$. Formally, $\gamma(c_f)$ is the minimum integer within $[0, d - 3]$ that makes:

$$l' \leq -s \sum_{i=\gamma(c_f)}^{d-1} \ln(1 - \frac{c_{f,i}}{sp}) \leq h'. \tag{17}$$

Notice that, when selecting a staring row $\gamma(c_f)$, we can ensure the estimation accuracy for flows whose sizes/spreads are within $[l' \cdot 2^{\gamma(c_f)}, h' \cdot 2^{\gamma(c_f)}]$. When increasing the starting row by 1, the accurate measurement range will be $[l' \cdot 2^{\gamma(c_f)+1}, h' \cdot 2^{\gamma(c_f)+1}]$. Apparently, these two intervals overlap when $2l' \leq h'$, which extends the estimation range to $[l' \cdot 2^{\gamma(c_f)}, h' \cdot 2^{\gamma(c_f)+1}]$.

At last, we formalize the parameter selection problem for SAS-LOG as in Equation 18. N' refers to the sum of virtual spreads when applying $\mathcal{FC}_{d,s}$, which can be obtained from historical data. The first constraint ensures that for each virtual element, its sampling probability p' will be within $(0, 1]$. The 2-4 constraints ensure that flows whose sizes/spreads are within $[l', 2^{d-3}h']$ satisfy the accuracy constraints. The last constraints ensure that the desired estimation range $[l, h]$ is within the estimation range.

$$\min M \quad \text{s.t.} \begin{cases} M \geq -\frac{N'}{\ln p} \\ 2l' \leq h' \\ p_\delta(d, s, p, l') \geq 1 - \epsilon \\ p_\delta(d, s, p, h') \geq 1 - \epsilon \\ l' \leq l < h \leq 2^{d-3}h' \end{cases} \tag{18}$$

VI. Experimental Results

In this section, we evaluate the performance of our algorithms through extensive experiments using real Internet traffic traces downloaded from CAIDA [23].

A. Experiment Setup

We conduct two sets of experiments to evaluate our algorithms' performance for per-flow spread estimation and per-flow size estimation, respectively. When performing per-flow spread estimation, we use five-minute data downloaded from CAIDA as the dataset, which has 513889 per-destination flows and 3150740 distinct elements. We use one-minute data downloaded from CAIDA as the dataset for per-flow size estimation, which contains 589740 per-source flows and 31259223 packets.

We run our evaluation on a server equipped with two six-core Intel Xeon E5-2643 v4 3.40GHz CPU and 256GB RAM. We have implemented our solutions SAS-LC and SAS-LOG in C++. For comparison purposes, we also implemented NDS [11] in C++. The hash functions used in our experiments are MURMUR3 hash with different initial seeds.

B. Memory Requirements

We compare SAS-LC, SAS-LOG, and NDS in terms of the on-chip memory they require to satisfy the constraints given in Section II-C. Table I and Table II show the memory requirements with respect to δ, ϵ, h, and l when applying three algorithms to per-flow spread estimation and per-flow size estimation. The required memory is computed according to [11] and this work.

Table I shows the memory requirements for spread estimation. For the setting of $\delta = 0.2, \epsilon = 0.1$ and $\delta = 0.1, \epsilon = 0.05$, we find out that SAS-LC is space-efficient, since it reduces around 10% the on-chip memory that NDS requires. Another observation is, due to the benefit of logarithmic compression, the memory that SAS-LOG requires is not sensitive to the value of h, making it more suitable for the per-flow size estimation where estimation range is large. As shown in Table II, for the setting of $\delta = 0.2, \epsilon = 0.1$ and $\delta = 0.1, \epsilon = 0.05$, SAS-LOG can save around 40% the on-chip memory that NDS requires, and SAS-LC can save around 10% the on-chip memory that NDS requires, depending on the values of h and l.

TABLE I: On-chip memory requirements of NDS, SAS-LC, and SAS-LOG for spread measurement (MB)

l	$\delta = 0.2, \epsilon = 0.1$						$\delta = 0.1, \epsilon = 0.05$					
	$h = 2000$			$h = 10000$			$h = 2000$			$h = 10000$		
	NDS	SAS-LC	SAS-LOG	NDS	SAS-LC	SAS-LOG	NDS	SAS-LC	SAS-LOG	NDS	SAS-LC	SAS-LOG
50	0.628	0.537	0.562	0.628	0.592	0.565	2.938	2.624	4.391	2.938	2.671	4.400
100	0.410	0.337	0.371	0.410	0.354	0.374	1.683	1.449	1.943	1.683	1.466	1.951
150	0.312	0.276	0.300	0.312	0.288	0.303	1.097	1.041	1.246	1.097	1.048	1.256
200	0.271	0.244	0.256	0.271	0.244	0.258	0.872	0.866	0.963	0.872	0.867	0.970
250	0.233	0.219	0.231	0.233	0.222	0.232	0.735	0.729	0.806	0.735	0.738	0.811
300	0.226	0.201	0.212	0.226	0.203	0.213	0.668	0.624	0.693	0.668	0.626	0.696

TABLE II: On-chip memory requirements of NDS, SAS-LC, and SAS-LOG for size measurement (MB)

l	$\delta = 0.2, \epsilon = 0.1$						$\delta = 0.1, \epsilon = 0.05$					
	$h = 50000$			$h = 200000$			$h = 50000$			$h = 200000$		
	NDS	SAS-LC	SAS-LOG	NDS	SAS-LC	SAS-LOG	NDS	SAS-LC	SAS-LOG	NDS	SAS-LC	SAS-LOG
100	4.067	3.087	2.236	4.067	3.510	2.237	16.700	12.660	11.918	16.700	14.130	11.919
200	2.688	2.113	1.644	2.688	2.386	1.644	8.650	7.368	6.488	8.650	8.048	6.491
300	2.244	1.752	1.430	2.244	1.969	1.430	6.629	5.279	4.947	6.629	5.833	4.949
400	1.895	1.559	1.309	1.895	1.746	1.310	5.376	4.437	4.102	5.376	4.852	4.105
500	1.688	1.437	1.233	1.688	1.598	1.233	4.539	3.794	3.601	4.539	4.120	3.604
1000	1.325	1.138	1.047	1.325	1.253	1.047	2.927	2.501	2.502	2.927	2.725	2.508

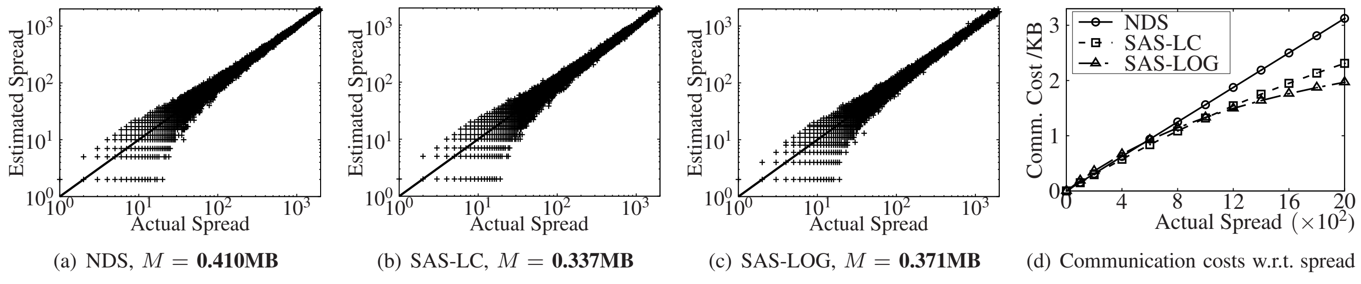


Fig. 7: Spread estimation accuracy of NDS, SAS-LC, and SAS-LOG when $\delta = 0.2, \epsilon = 0.1, h = 2000, l = 100$

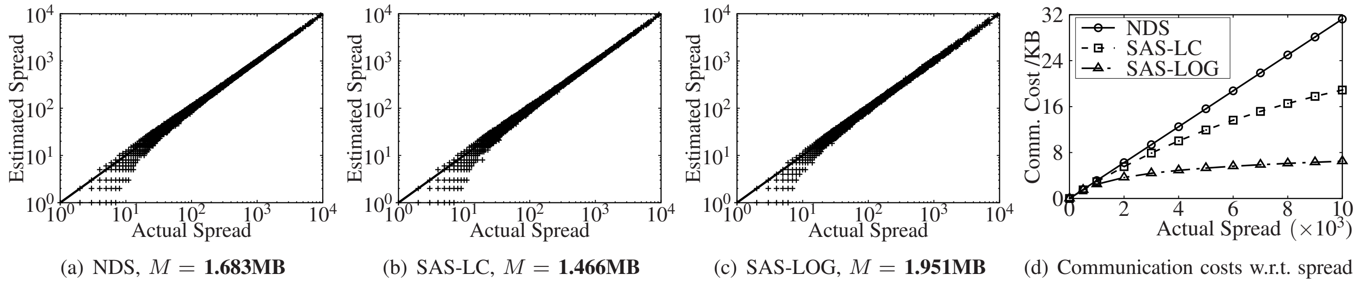
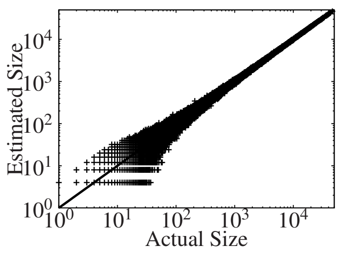


Fig. 8: Spread estimation accuracy of NDS, SAS-LC, and SAS-LOG when $\delta = 0.1, \epsilon = 0.05, h = 10000, l = 100$

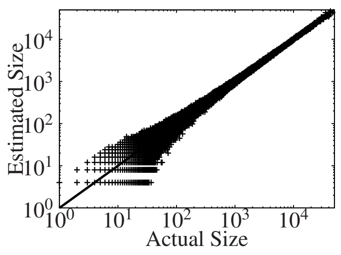
C. Estimation accuracy

1) Experiments on per-flow spread estimation: We evaluate the estimation accuracy of NDS, SAS-LC, and SAS-LOG under two sets of constraints, respectively $\delta = 0.2, \epsilon = 0.1, h = 2000, l = 100$ and $\delta = 0.1, \epsilon = 0.05, h = 10000, l = 100$. For each algorithm, its on-chip memory size is set to the minimum value satisfying the given constraints. The other parameters are set to the optimal parameters according to [11] and this work. Fig. 7(a) - 7(c) show the estimation results of three algorithms under the first set of constraints, where the on-chip memory sizes of NDS, SAS-LC, and SAS-LOG are 0.410MB, 0.337MB, and 0.371MB. In three plots, the x-axis denotes the actual spread, the y-axis represents the estimated spread, and each point refers to a flow. The more close a point is to the line $y = x$, the more accurate this estimation is.

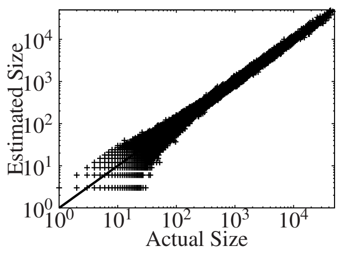
We find out that NDS's estimation accuracy grows as flow spread increases, even though such high estimation accuracy for large flows is beyond the requirement. In Table III, we show the actual relative error bounds for different flows. For flows whose spreads are within $[100, 2000]$, the actual relative error bounds (with probability 90%) are all under the constraint $\delta = 0.2$. In other words, by using self-adaptive sampling and configuring lower sampling rates for large flows, SAS-LC and SAS-LOG show slightly worse accuracy on large flows than NDS does, but still manage to satisfy the given constraints. Moreover, self-adaptive sampling can help in reducing both on-chip memory usage and communication cost. Compared to NDS, SAS-LC and SAS-LOG save 17.8% and 9.5% of the on-chip memory usage and reduce up to 25.8% and 35.5% communication cost for large flows (when



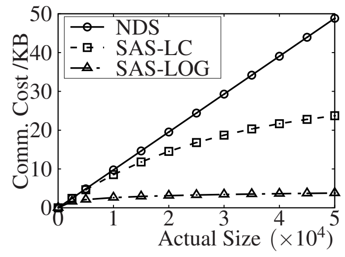
(a) NDS, $M =$ **2.688MB**



(b) SAS-LC, $M =$ **2.113MB**

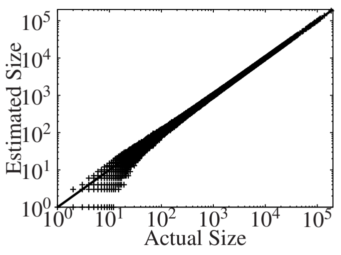


(c) SAS-LOG, $M =$ **1.644MB**

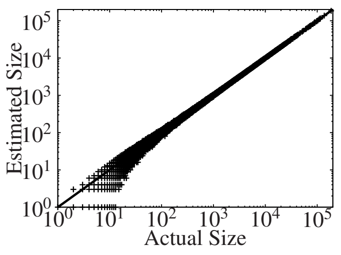


(d) Communication costs w.r.t. size

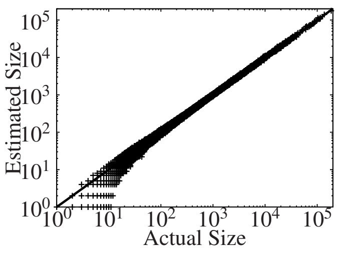
Fig. 9: Size estimation accuracy of NDS, SAS-LC, and SAS-LOG when $\delta = 0.2, \epsilon = 0.1, h = 50000, l = 200$



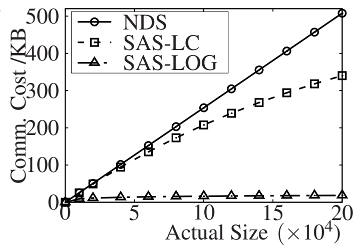
(a) NDS, $M =$ **8.650MB**



(b) SAS-LC, $M =$ **8.048MB**



(c) SAS-LOG, $M =$ **6.491MB**



(d) Communication costs w.r.t. size

Fig. 10: Size estimation accuracy of NDS, SAS-LC, and SAS-LOG when $\delta = 0.1, \epsilon = 0.05, h = 200000, l = 200$

flow spread is 2000). We can observe similar results from the experimental results under the second set of constraints (as shown in Fig. 8 and Table IV).

TABLE III: Actual relative error bound for spread estimation ($\delta = 0.2, \epsilon = 0.1, h = 2000, l = 100$)

spread / algorithm	all flows	$1 \sim 100$	$100 \sim 500$	$500 \sim 1000$	$1000 \sim 2000$
NDS	1.500	1.500	0.153	0.077	0.062
SAS-LC	1.503	1.503	0.172	0.114	0.096
SAS-LOG	1.203	1.203	0.168	0.124	0.140

TABLE IV: Actual relative error bound for spread estimation ($\delta = 0.1, \epsilon = 0.05, h = 10000, l = 100$)

spread / algorithm	all flows	$1 \sim 100$	$100 \sim 1000$	$1000 \sim 5000$	$5000 \sim 10000$
NDS	1.000	1.000	0.075	0.028	0.013
SAS-LC	1.000	1.000	0.079	0.038	0.043
SAS-LOG	1.000	1.000	0.088	0.098	0.097

2) Experiments on per-flow size estimation: We use two sets of constraints, respectively $\delta = 0.2, \epsilon = 0.1, h = 50000, l = 200$ and $\delta = 0.1, \epsilon = 0.05, h = 200000, l = 200$, to evaluate the estimation accuracy for per-flow size estimation. Fig. 9 and Table V show the results under the first set of constraints. Fig. 10 and Table VI show the results under the second set of constraints. Recall that, SAS-LOG uses a logarithmic compression function to meet the need of measuring large flows, which can achieve high efficiency in per-flow size measurement. For example, under the first set of constraints, SAS-LC and SAS-LOG reduce the on-chip memory usage of NDS by 11.8% and 38.8%. Besides, they reduce the communication cost of NDS up to 51.4% and 92.2% when the flow size is 50000. Clearly, our solution SAS-LOG is the winner for per-flow size measurement.

TABLE V: Actual relative error bound for size estimation ($\delta = 0.2, \epsilon = 0.1, h = 50000, l = 200$)

size / algorithm	all flows	$1 \sim 200$	$200 \sim 1000$	$1000 \sim 10000$	$10000 \sim 50000$
NDS	1.020	1.020	0.155	0.064	0.023
SAS-LC	1.054	1.054	0.158	0.075	0.062
SAS-LOG	1.207	1.312	0.176	0.176	0.171

TABLE VI: Actual relative error bound for size estimation ($\delta = 0.1, \epsilon = 0.05, h = 200000, l = 200$)

size / algorithm	all flows	$1 \sim 200$	$200 \sim 1000$	$1000 \sim 50000$	$50000 \sim 200000$
NDS	1.000	1.000	0.078	0.034	0.005
SAS-LC	1.000	1.000	0.077	0.033	0.016
SAS-LOG	1.000	1.000	0.082	0.079	0.099

VII. Conclusion

This paper proposes an efficient self-adaptive sampling framework for network traffic measurement, which works for both per-flow size estimation and per-flow spread estimation. Based on two different compression functions, we present two algorithms, SAS-LC and SAS-LOG, to sample each flow's (distinct) elements with a probability adapted to flow size/spread, which save on-chip space and reduce communication cost while ensuring estimation accuracy. The experimental results based on real Internet traffic traces demonstrate that our solutions can be flexibly configured to meet the measurement interests of different applications and work efficiently with small on-chip memory and small communication overhead.

Acknowledgments

This research was supported by the National Natural Science Foundation of China (NSFC) (Grant No. 62072322, 61873177, U20A20182), and Natural Science Foundation (Grant No. CNS-1719222).

REFERENCES

[1] N. Duffield, C. Lund, and M. Thorup, "Learn more, sample less: control of volume and variance in network measurement," *IEEE Transactions on Information Theory*, vol. 51, no. 5, pp. 1756–1775, 2005.

[2] J. Zheng, H. Xu, G. Chen, and H. Dai, "Minimizing transient congestion during network update in data centers," in *Proc. of IEEE International Conference on Network Protocols (ICNP 2015)*, 2015, pp. 1–10.

[3] H. Xu, Z. Yu, C. Qian, X. Li, Z. Liu, and L. Huang, "Minimizing flow statistics collection cost using wildcard-based requests in SDNs," *IEEE/ACM Transactions on Networking*, vol. 25, no. 6, pp. 3587–3601, 2017.

[4] T. Li, S. Chen, W. Luo, and M. Zhang, "Scan detection in high-speed networks based on optimal dynamic bit sharing," in *Proc. of the IEEE Conference on Computer Communications (INFOCOM 2011)*, 2011, pp. 3200–3208.

[5] Y. Li, H. Wu, T. Pan, H. Dai, J. Lu, and B. Liu, "CASE: cache-assisted stretchable estimator for high speed per-flow measurement," in *Proc. of the IEEE Conference on Computer Communications (INFOCOM 2016)*, 2016, pp. 1–9.

[6] H. Alasmary, A. Abusnaina, R. Jang, M. Abuhamad, A. Anwar, D. NYANG, and D. Mohaisen, "Soteria: Detecting adversarial examples in control flow graph-based malware classifiers," in *Proc. of the IEEE International Conference on Distributed Computing Systems (ICDCS 2020)*, 2020, pp. 1296–1305.

[7] A. Abusnaina, R. Jang, A. Khormali, D. Nyang, and D. Mohaisen, "Dfd: Adversarial learning-based approach to defend against website fingerprinting," in *Proc. of the IEEE Conference on Computer Communications (INFOCOM 2020)*, 2020, pp. 2459–2468.

[8] A. Kumar, J. Xu, and J. Wang, "Space-code bloom filter for efficient per-flow traffic measurement," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 12, pp. 2327–2339, 2006.

[9] C. Hu, B. Liu, S. Wang, J. Tian, Y. Cheng, and Y. Chen, "ANLS: adaptive non-linear sampling method for accurate flow size measurement," *IEEE Transactions on Communications*, vol. 60, no. 3, pp. 789–798, 2012.

[10] F. Hao, M. Kodialam, and T. Lakshman, "ACCEL-RATE: a faster mechanism for memory efficient per-flow traffic estimation," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 32, no. 1, 2004, pp. 155–166.

[11] Y. Sun, H. Huang, C. Ma, S. Chen, Y. Du, and Q. Xiao, "Online spread estimation with non-duplicate sampling," in *Proc. of the IEEE Conference on Computer Communications (INFOCOM 2020)*, 2020, pp. 2440–2448.

[12] P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier, "Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm," in *Discrete Mathematics and Theoretical Computer Science*, 2007, pp. 137–156.

[13] S. Heule, M. Nunkesser, and A. Hall, "Hyperloglog in practice: algorithmic engineering of a state of the art cardinality estimation algorithm," in *Proc. of the 16th International Conference on Extending Database Technology (EDBT 2013)*, 2013, pp. 683–692.

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

[15] R. Jang, S. Moon, Y. Noh, A. Mohaisen, and D. Nyang, "Instameasure: Instant per-flow detection using large in-dram working set of active flows," in *Proc. of the IEEE International Conference on Distributed Computing Systems (ICDCS 2019)*, 2019, pp. 2047–2056.

[16] R. Jang, D. Min, S. Moon, D. Mohaisen, and D. Nyang, "Sketchflow: Per-flow systematic sampling using sketch saturation event," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 1339–1348.

[17] P. Lieven and B. Scheuermann, "High-speed per-flow traffic measurement with probabilistic multiplicity counting," in *Proc. of the IEEE Conference on Computer Communications (INFOCOM 2010)*, 2010, pp. 1–9.

[18] M. Yoon, T. Li, S. Chen, and J. kwon Peir, "Fit a spread estimator in small memory," in *Proc. of the IEEE Conference on Computer Communications (INFOCOM 2009)*, 2009, pp. 504–512.

[19] Y. Lu, A. Montanari, B. Prabhakar, S. Dharmapurikar, and A. Kabbani, "Counter braids: a novel counter architecture for per-flow measurement," *ACM SIGMETRICS Performance Evaluation Review*, vol. 36, no. 1, pp. 121–132, 2008.

[20] Y. Zhou, Y. Zhou, M. Chen, Q. Xiao, and S. Chen, "Highly compact virtual counters for per-flow traffic measurement through register sharing," in *Proc. of the IEEE GLOBECOM 2016*, 2016, pp. 1–6.

[21] Y. Zhou, Y. Zhou, S. Chen, and Y. Zhang, "Per-flow counting for big network data stream over sliding windows," in *Proc. of the IEEE/ACM IWQoS 2017*, 2017, pp. 1–10.

[22] ——, "Highly compact virtual active counters for per-flow traffic measurement," in *Proc. of the IEEE Conference on Computer Communications (INFOCOM 2018)*, 2018.

[23] CAIDA, "The CAIDA UCSD Anonymized Internet Traces 2016," http://www.caida.org/data/passive/passive_2016_dataset.xml, accessed July 28, 2019.

[24] T. Yang, J. Xu, X. Liu, P. Liu, L. Wang, J. Bi, and X. Li, "A generic technique for sketches to adapt to different counting ranges," in *Proc. of the IEEE Conference on Computer Communications (INFOCOM 2019)*, 2019, pp. 2017–2025.

[25] M. Yoon, T. Li, S. Chen, and J.-K. Peir, "Fit a compact spread estimator in small high-speed memory," *IEEE/ACM Transactions on Networking (TON)*, vol. 19, no. 5, pp. 1253–1264, 2011.

[26] H. Huang, Y. Sun, S. Chen, S. Tang, K. Han, J. Yuan, and W. Yang, "You can drop but you can't hide:k-persistent spread estimation in high-speed networks," in *Proc. of the IEEE Conference on Computer Communications (INFOCOM 2018)*, 2018, pp. 1889–1897.

[27] Y. Zhou, Y. Zhou, S. Chen, and Y. Zhang, "Highly compact virtual active counters for per-flow traffic measurement," in *Proc. of the IEEE Conference on Computer Communications (INFOCOM 2018)*, 2018, pp. 1–9.

[28] Y. Zhou, Y. Zhou, M. Chen, and S. Chen, "Persistent spread measurement for big network data based on register intersection," in *Proc. of the ACM on Measurement and Analysis of Computing Systems*, vol. 1, no. 1. ACM, 2017, p. 15.

[29] N. Duffield, C. Lund, M. Thorup, and M. Thorup, "Flow sampling under hard resource constraints," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 32, no. 1, 2004, pp. 85–96.

[30] S. L. Feibish, Y. Afek, A. Bremler-Barr, E. Cohen, and M. Shagam, "Mitigating DNS random subdomain DDoS attacks by distinct heavy hitters sketches," in *Proc. of the fifth ACM/IEEE Workshop on Hot Topics in Web Systems and Technologies*, 2017, p. 8.

[31] V. Braverman, E. Grigorescu, H. Lang, D. P. Woodruff, and S. Zhou, "Nearly optimal distinct elements and heavy hitters on sliding windows," in *APPROX-RANDOM 2018*, 2018, pp. 1–22.

[32] X. Dimitropoulos, P. Hurley, and A. Kind, "Probabilistic lossy counting: An efficient algorithm for finding heavy hitters," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 1, pp. 7–16, 2008.

[33] Y. Zhang, S. Singh, S. Sen, N. Duffield, and C. Lund, "Online identification of hierarchical heavy hitters: Algorithms, evaluation, and application," *Proc. of ACM SIGCOMM IMC*, pp. 101–114, October 2004.

[34] Y. Zhou, Y. Zhang, C. Ma, S. Chen, and O. O. Odegbile, "Generalized sketch families for network traffic measurement," *Proc. of the ACM on Measurement and Analysis of Computing Systems*, vol. 3, no. 3, pp. 1–34, 2019.

[35] C. Hu, B. Liu, H. Zhao, K. Chen, Y. Chen, Y. Cheng, and H. Wu, "Discount counting for fast flow statistics on flow size and flow volume," *IEEE/ACM Transactions on Networking*, vol. 22, no. 3, pp. 970–981, 2014.

[36] H. Dai, M. Shahzad, A. X. Liu, M. Li, Y. Zhong, and G. Chen, "Identifying and estimating persistent items in data streams," *IEEE/ACM Transactions on Networking*, vol. 26, no. 6, pp. 2429–2442, 2018.

[37] Y. Zhang, "An adaptive flow counting method for anomaly detection in SDN," in *Proc. of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*. New York, NY, USA: Association for Computing Machinery, 2013, pp. 25–30.

[38] G. Cheng and J. Yu, "Adaptive sampling for openflow network measurement methods," in *Proc. of the 12th International Conference on Future Internet Technologies*. New York, NY, USA: Association for Computing Machinery, 2017, pp. 1–7.

[39] K.-Y. 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, 1990.

[40] C. Estan, G. Varghese, and M. Fisk, "Bitmap algorithms for counting active flows on high speed links," in *Proc. of the 3rd ACM SIGCOMM conference on Internet measurement*, 2003, pp. 153–166.