Virtual Filter for Non-duplicate Sampling

Chaoyi Ma* Haibo Wang* Olufemi O Odegbile Shigang Chen
Department of Computer & Information Science & Engineering,
University of Florida, Gainesville, FL 32611, USA,
Email: {ch.ma, wanghaibo, oodegbile}@ufl.edu, sgchen@cise.ufl.edu

***Abstract*—Sampling is key to handling mismatch between the line rate and the throughput of a network traffic measurement module. Flow-spread measurement requires non-duplicate sampling, which only samples the elements (carried in packet header or payload) in each flow when they appear for the first time and blocks them for subsequent appearances. The only prior work for non-duplicate sampling incurs considerable overhead, and has two practical limitations: It lacks a mechanism to set an appropriate sampling probability under dynamic traffic conditions, and it cannot efficiently handle multiple concurrent sampling tasks. This paper proposes a *virtual filter* design for non-duplicate sampling, which reduces the processing overhead by about half and reduces the memory overhead by an order of magnitude or more under some practical settings. It has a mechanism to automatically adapt its sampling probability to the traffic dynamics. It can be extended to solve a new problem called *non-duplicate distribution sampling*, which samples packets based on a probability distribution to support multiple concurrent measurement tasks.**

I. Introduction

Traffic measurement is a fundamental function that provides crucial information about communication activities and network states for an array of core network functions such as traffic engineering, resource provision, threat monitoring, adaptive routing decision [1], [2], [3]. The widely used tools, including NetFlow [4] and sFlow [5], employ sampling to deal with mismatch between the packet forwarding line rate and the throughput of the traffic measurement module at a router. The reason for the rate mismatch is that packet forwarding, as the key function of a router, is given top priority in resource allocation (e.g., processing circuitry and on-die memory), while the traffic measurement module, as a supporting function, is of lower priority.

NetFlow and sFlow measure per-flow statistics such as *flow size*, i.e., the number of packets in each flow. Many sketches have also been proposed to measure flow size with better memory efficiency, including NitroSketch [6], Elastic Sketch [7], SketchLearn [8], SketchVisor [9], UnivMon [10] and many others [11], [12], [13], [14]. When there is a mismatch between the line rate and the throughput of a flow-size measurement module, we simply sample each packet independently with a certain probability p and only forward the sampled packets to the measurement module. Sampling can be easily implemented by taking a random number r from a certain range $[0, N)$, and a packet is sampled for further processing if $r \leq pN$. This approach is stateless, with negligible memory overhead.

Flow Spread and Non-duplicate Sampling: However, more sophisticated traffic measurement will require sampling to be done differently. Consider the problem of measuring the *flow spread*, which is the number of *distinct elements* in each flow [15], [16], [17], [18], [19], [20], where elements may be chosen from the packet-header fields or payload based on application need. With the flow spread information, we can identify super spreaders [21], [22], [23], [24], [25] or detect malicious activities [26], [27], [28]. As an example, we may define a flow as all packets to a certain destination address, and define the element to be measured as the source address of each packet. The spread of a flow is the number of distinct sources that have contacted the same destination. A flow with unusually large spread signals crowd flush or DDoS attack, either of which requires immediate attention from the system admin team.

The uniqueness of spread measurement is that each distinct element in a flow is counted only once regardless of the number of occurrences. That is, duplicates in the flow should be removed. If there is a mismatch between the line rate and the throughput of a flow-spread measurement module, we need *non-duplicate sampling*, which is defined as follows: *If a packet carries an element that appears in the flow for the first time, we sample it with a probability p; if a packet carries an element that has appeared in the flow before, we ignore it.*

Challenge and Prior Art: To implement non-duplicate sampling, the key is to determine whether a received element (carried in a received packet) is new or has been seen before. A Bloom filter [29] is easy to come in mind, which records all received elements in a bit array and checks whether a newly received one has already been recorded. However, using a Bloom filter is too expensive both in processing overhead and in memory usage. Each received element requires multiple hash operations and takes multiple bits to record. This overhead happens on the packet-forwarding path of the data plane where it is highly desired to keep processing as simple as possible and keep on-die memory footprint as small as possible.

The only prior work on non-duplicate sampling for traffic measurement is a recent two-phase protocol [30]. Much more efficient than a Bloom filter, it requires two hashes and uses one bit to record each received element. However, its design does not have a mechanism to handle dynamic traffic conditions in real time, and therefore its performance will degrade

*Co-first authors

978-1-6654-4131-5/21/$31.00 ©2021 IEEE

as traffic deviates from what its setting expects. It cannot efficiently handle multiple sampling tasks and has to deal with them individually, causing overhead to multiply. Moveover, its sampling performance has significant room for improvement: First, two hashes per packet are more expensive than generating a random number in traditional packet sampling. The reason is that hashes in the two-phase protocol are required to have good randomness in their outputs and therefore such a hash could be used for generating a random number. Second, the memory overhead of the two-phase protocol can be very significant if there is a very large number of elements to be recorded, as our experiments will show.

Contributions: First, this paper proposes a new *virtual filter algorithm* that implements non-duplicate sampling with one hash per packet and records only a fraction of the received elements, with much smaller memory footprint than [30], especially when the sampling probability is small. We prove that our algorithm correctly implements non-duplicate filtering. We formally derive the optimal parameters that minimize the memory requirement under any given sampling probability. We also design a new mechanism for the virtual filter to adapt its sampling probability automatically in real time under dynamic traffic conditions.

Second, we extend the virtual filter algorithm to solve a new problem that has not been investigated before: *non-duplicate distribution sampling*, which performs non-duplicate sampling on packets not based on a single probability but instead based on a probability distribution defined by a number of probability values. With the same processing overhead, distribution sampling produces multiple output streams, each corresponding to a different sampling probability, feeding to multiple traffic measurement modules, some of which may be interested only in large-spread flows (small sampling probabilities), while other may be interested in broad-scoped measurement (large sampling probabilities).

Third, we implement the new sampling algorithm and evaluate it through trace-based experiments using real-world packet streams. The experimental results show that the new algorithm can operate at a line rate much higher than the prior two-phase protocol, while using much smaller memory, oftentimes, an order of magnitude smaller. We also perform a case study of using non-duplicate sampling to support flow spread measurement. It greatly improves the measurement throughput and surprisingly also improves measurement accuracy even when less memory is allocated.

II. Preliminaries

A. Problem Statement

We make the problem statement based on a generic data stream model for general applicability. A data stream is a continuous sequence of data items. Each item x may appear in the stream for an arbitrary number of times, resulting in duplicates. We will show how to map packets to data items in this model shortly.

The problem of *non-duplicate sampling* is defined as follows: Given a sampling probability p, for the next received item x, if it is the first time that x shows up in the stream, we output x with probability p; otherwise, we ignore it.

Any algorithm that solves the above problem will need a data structure to remember the data items that have been seen. Any data structure will have a limited capacity: the expected number of distinct items that it can record is determined by the amount of memory allocated. We define a *sampling period* as the expected number n of distinct items that an algorithm can process before its data structure is so saturated that it can no longer ensure non-duplicate sampling. After a period, we will have to start a new period and initialize the data structure. Therefore, non-duplicate sampling is achieved for data stream within each period.

Beside correctness, the performance of a non-duplicate sampling algorithm will be judged by three metrics: (1) Given a period n, it should use as little on-die memory as possible; (2) given a memory allocation, it should work for a period as long as possible; (3) its processing overhead per item should be as little as possible, so as to support a line rate as large as possible.

B. Spread Measurement

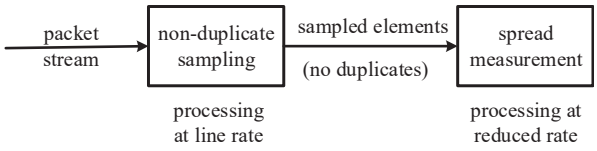


Fig. 1: System Model

We will apply non-duplicate sampling on network traffic measurement, as illustrated in Figure 1. A non-duplicate sampling module processes the arrival packet stream at line rate. Its output, which is a sub-stream of sampled packets, is sent to a traffic measurement module for spread measurement at a reduced rate that the module can handle.

We can model network traffic as a data stream. Each packet is abstracted as a data item $x = \langle f, e \rangle$, where f is a flow label and e is an element. We define a flow f as the set of packets that carry the same flow label f, which may be TCP flow identifier, source address (for per-source flow), destination address (for per-destination flow), destination address/port (for per-service flow), URL (for per-content flow considering http traffic only), etc. We define an element e as a value or a value combination from the packet headers or the payload. Take a few examples: For per-source flows, we may measure the number of distinct destination addresses in each flow, which helps us track network reconnaissance activities, worm-infected hosts, botnet communications and malicious scanners [27], [28]. For per-destination flows, we may measure the number of distinct source addresses in each flow, which helps us track potential botnet-based denial-of-service or denial-of-quality attacks, service hotspots, or congested network activities [26], [31]. For per-URL flows, we may measure the number of distinct source/port pairs, which shows the interest in the content across the Internet.

We stress that, based on the generic data streaming model, our non-duplicate sampling algorithm has broader applications

beyond the networking area. For example, consider an Internet search engine and the stream of search requests (data items) that it receives. We may use the new algorithm from this paper to filter duplicate searches. In another example, consider an e-commerce company and the web visits to its products. We may use the algorithm to filter repeated visits of the same product by the same user.

III. Non-duplicate Sampling and Virtual Filter

We begin with a naive approach based on Bloom filter and then move to the existing two-phase protocol, which helps motivate for our new solution.

A. Sampling with Bloom Filter

A Bloom filter is a bitmap B of m bits, with two operations.

• *Recording:* We record an item x by hashing x to d bit indexes, $H_i(x) \in [0, m)$, $0 \leq i < d$, and setting those bits to ones, i.e., $B[H_i(x)] = 1$.

• *Look-up:* Given a data item x, we check whether the d bits, $B[H_i(x)]$, $0 \leq i < d$, are all ones. If so, we claim that x is in the filter; otherwise, we claim that x is not in the filter.

Each time when we receive a data item x, we first look up in B to see if it is already recorded. If so, we ignore x. Otherwise, we record x in B and pass x through as output. This approach makes sure that any data item can pass the filter only once and there will be no duplicate in the items that have passed through. However, a Bloom filter has false positives, which means that some data items may not pass the filter even for their first appearances. The probability P_{fp} of false positive increases as we record more and more items in the filter — essentially, the sampling probability, $1-P_{fp}$, changes over time. It does not enforce a given, constant sampling probability. Moreover, a Bloom filter has other disadvantages: (1) Each arrival data item requires d hashes and $O(d)$ memory accesses (read and write); (2) it takes d bits to record an item for duplicate filtering.

B. Two-phase Protocol (TP)

Sun et al. proposed a two-phase protocol (TP) for non-duplicate sampling [30]. It also uses a bitmap B of m bits but records every received item x by setting a single bit, $B[h(x)]$, to one, where $h(x) = H(x) \mod m$ and $H(x)$ is a uniform hash function whose range is larger than m. More specifically, each time when it receives a data item x, its first phase is a traditional packet sampling of probability $\frac{m}{z}p$, where z is the current number of zeros in the bitmap. The second phase checks if $B[h(x)]$ is one. If so, we ignore the item; otherwise, we set $B[h(x)] = 1$ and pass x through the second phase. TP outputs x if it passes both phases.

For any item x that appears for the first time, the probability for it to be sampled in the first phase is $\frac{m}{z}p$, and the probability to find $B[h(x)] = 0$ in the second phase is $\frac{z}{m}$. Therefore, the probability for the item to pass through as output is $\frac{m}{z}p \times \frac{z}{m} = p$. For any item x that appears for additional times, because $B[h(x)] = 1$, those appearances will be ignored.

TP records each item by setting one bit. While this is more memory-efficient than a Bloom filter, it will still take a large amount of space over an extended sampling period. We may look at this issue from a different angle: Suppose that we are given a fixed memory allocation of m bits. One bit per item allows a sampling period to contain at most m distinct items. If a better sampling algorithm somehow only requires to record a small percentage, say 10%, of all items that have been seen, then the period can be enlarged 10 folds, containing up to $10m$ distinct items, before the m-bit memory is exhausted. Such an algorithm will be able to perform non-duplicate sampling for a much larger data stream, for example, up to $10m$ distinct items instead m items by TP in the above example.

Moreover, TP takes two hashes to process each data item, one in each phase. If we can reduce that number to one, we can potentially double the line rate that the sampling module can maximally support.

TP lacks a mechanism to automatically adapt to the evolving traffic dynamics in real time, which is serious practical limitation.

Finally TP does not consider non-duplicate distribution sampling that operates with multiple sampling probabilities at the same overhead, which our algorithm will consider later.

C. Virtual Filter (VF)

We now present virtual filter algorithm (VF) for non-duplicate sampling which performs exactly one hash per data item and records only a fraction of all data items in its memory. The operation of our virtual filter algorithm is simple but we stress that this is an advantage, as the sampling module that processes packet stream at line rate cannot afford complicated computations.

• **Data Structure and Algorithm**: The main data structure is a *virtual filter*, which is a bitmap B of m' bits, but only its first m bits are real. We call $B[0]...B[m-1]$ the real part of the filter and $B[m]...B[m'-1]$ the virtual part of the filter. Each time when we receive a data item x, we perform hash $h(x) = H(x) \mod m'$, where $H(x)$ is a hash function whose range is larger than m'. We do the following three steps:

Step 1: If $h(x) \geq m$, it falls in the virtual part of the filter, we ignore the data item, which does not cause any memory overhead or any further processing overhead as recording does not happen for this item. If $h(x) < m$, it falls in the real part of the filter and we continue with the next step.

Step 2: If $B[h(x)]$ is one, we do nothing further and the item is blocked; otherwise, set $B[h(x)] = 1$ and move to the next step.

Step 3: If $h(x) < \frac{mm'p}{z}$ where z is the number of zeros in the real part of the filter before x is recorded, we pass x through as output; otherwise, we block the item.

Step 1 is designed to avoid having to record every item received, so as to save memory space. Step 2 is to filter duplicates. Its sampling rate however changes over time as the bits in the filter are set to ones. Step 3 is designed to counter the rate change in Step 2 so that the overall sampling rate remains the same over time. We will show that sampling is actually performed at all steps, though for different purposes. The trick is to implement them with a single hash operation

under progressive conditional probabilities, which together ensure non-duplicate sampling with memory and processing efficiencies.

Step 1 performs sampling with probability $\frac{m}{m'}$. Only when item x is hashed to the first m bits in the real part of the filter, it passes onto Step 2. Otherwise, the item is ignored. Therefore, a fraction $\frac{m'-m}{m'}$ of all distinct items will never be recorded, which saves memory, in contrast to TP's recording of all items.

Under the condition that item x passes the previous step, Step 2 checks the bit that x is hashed to. Even if x appears for the first time, it may be hashed to a bit that is already set to one by another item. In this case, x will be blocked. Only when the bit is zero, x passes Step 2 and the bit is set to one. Therefore, Step 2 does sampling too, with a probability that decreases over time as fewer and fewer bits in the real part remain zeros. The purpose of Step 2 is to filter duplicates since subsequent appearances of x will all be hashed to the same bit that is one. Its sampling with *decreasing* probability is a by-product of the filtering design. We need to deal with it in Step 3.

Under the condition that item x passes the first two steps, Step 3 performs another sampling, with a probability that increases over time to compensate the sampling probability that decreases over time in Step 2. Because $h(x) < m$ after passing Step 2, this probability is $\frac{\frac{mm'p}{z}}{m} = \frac{m'p}{z}$, which increases over time because z decreases over time as more bits in the real part are set to ones by new arrival items. The choice of such a probability is by design to make sure that when we combine the three samplings over the three steps, the final sampling probability is exactly p for any item when it is received for the first time. This will be proved shortly.

We know that $z \leq m$ because the number of zeros in the real part cannot be more than the number of bits there. The value of z starts at m and decreases as bits in the real part are set to ones, which in turn causes the bound $\frac{mm'p}{z}$ in Step 3 increases. Since $h(x) < m$ after passing Step 1, for Step 3 to perform sampling, the bound should satisfy $\frac{mm'p}{z} < m$. The current sampling period will terminate when $\frac{mm'p}{z} = m$, i.e., $z = m'p$. This termination condition is needed for the correctness of our sampling algorithm and for the proof of the theorems (to be given). Because $z \leq m$, it implies a constraint that $m \geq m'p$ when we set the optimal values of m and m'. The pseudo-code of the algorithm is given in Alg. 1.

• **Example**: Consider a data stream in Fig. 2, whose first 10 distinct data items are x_1 through x_{10} with x_2 appearing twice. The virtual filter consists of m bits in its real part and $m' - m$ bits in its virtual part. Suppose $p = 0.1$. We set $\frac{m}{m'} = \frac{1}{pe} \approx 3.6$, which is optimal as our analytical results will show shortly. That means the size of the virtual part is almost 2.6 times that of the real part. As items are hashed to the bits in the filter uniformly at random in Step 1, about 28% of them are hashed to the real part and 78% to the virtual part. In this example, suppose that x_2, x_5 and x_6 are hashed to the real part and other items are hashed to the virtual part. Below we walk through the example, item by item.

Algorithm 1 Non-duplicate sampling using VF with probability p

Input: sampling probability p, number of distinct items tend to process in a period: n, data stream
Action: perform non-duplicate sampling
// setting m, m' according to Theorem 2
if $p < \frac{1}{e}$ **then**
  $m = npe, m' = n$
else
  $m = -\frac{n}{\ln p}, m' = -\frac{n}{\ln p}$
create a bitmap B of m bits, set $z = m$
for each data item x **do**
  $i = h(x) = H(x) \mod m'$
  if $i < m$ **then**
    if $B[i] = 0$ **then**
      $B[i] = 1$
      if $i < \frac{mm'p}{z}$ **then**
        x is sampled
      $z = z - 1$
  if $z \leq m'p$ **then**
    break //end the period

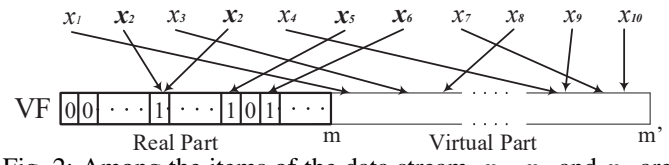


Fig. 2: Among the items of the data stream, x_2, x_5 and x_6 are hashed to the real part of the filter. Other items are hashed to the virtual part and thus blocked. The second appearance of x_2 is blocked by Step 2 because the bit it hashes to has been set to one by the first appearance of x_2. The condition in Step 3 means that only some of the items hashed to the real part can pass this step.

When x_1 arrives, it is hashed in Step 1 to the virtual part and is thus ignored without incurring further overhead. When x_2 arrives for the first time, it is hashed in Step 1 to a bit in the real part. The bit is set in Step 2 from zero to one. Suppose it passes the condition in Step 3. Then it passes the whole filter. When x_3 arrives, it is hashed in Step 1 to the virtual part.

When x_2 arrives for the second time, it is hashed in Step 1 to the same bit that it was hashed to before. That bit is already one, and thus the item is blocked.

For the rest of the stream, item x_4 is hashed to the virtual part; x_5 and x_6 are hashed to the real part, setting their bits to ones but failing the condition in Step 3 to pass the filter; items x_7 through x_{10} are hashed to the virtual part. In the end, only x_2 passes the filter when it appears for the first time.

• **Correctness Proof, Setting Optimal Parameter, and Performance Comparison**: For correctness, any data item will pass the filter with probability p at its first appearance and will be blocked for subsequent appearances, which is proved in Theorem 1. For optimal parameter setting, given

the length of a sampling period (which is specified as the expected number n of distinct items in the period), we show what the minimum memory m' is and how to set the value of m in Theorem 2. Given an allocated memory m, we show that what the maximum sampling period will be in Theorem 3. The issue of setting the sampling probability under dynamic traffic conditions will be addressed later in Section III-D. For performance, VF only requires one hash operation to process each data item. The average number of memory accesses made for each data item includes $\frac{m}{m'}$ reads and $\frac{m}{m'}$ writes, which are both smaller than one as many items are hashed to the virtual part of the filter and do not incur any actual memory access. We compare VF with TP in Corollaries 1 and 2, which show that VF will never perform worse than TP. It performs better than the latter when $p \leq \frac{1}{e}$, and the gap increases when p decreases. Our numerical analysis demonstrates very significant improvement when p is small.

Theorem 1: Any data item passes the filter with probability p at its first appearance. It is blocked for further appearances.

Proof: For any item x that appears for the first time, the probability for it to move through Step 1 to Step 2 is $\frac{m}{m'}$. The probability for it to move through Step 2 to Step 3 is $\frac{z}{m}$. For Step 3, because we already know that $h(x) < m$ as it passes Step 1, the probability to pass Step 3 is $\frac{\frac{mm'p}{z}}{m} = \frac{m'p}{z}$. Hence, the probability for x to pass through the filter is $\frac{m}{m'} \times \frac{z}{m} \times \frac{m'p}{z} = p$. For any item x that appears for additional times, it will be blocked in the second step as $B[h(x)] = 1$. Therefore, all those appearances will not pass the filter. ■

Theorem 2: Let n be the expected number of distinct items to be processed in each sampling period. The optimal parameter setting of VF is

$$m' = \begin{cases} n, & p < \frac{1}{e} \\ -\frac{n}{\ln p}, & \frac{1}{e} \leq p < 1 \end{cases} \quad m = \begin{cases} npe, & p < \frac{1}{e} \\ -\frac{n}{\ln p}, & \frac{1}{e} \leq p < 1 \end{cases} \tag{1}$$

which minimizes the size m for the real part of the filter, under a given non-duplicate sampling probability p.

Proof: Among the n distinct data items, the expected number of items recorded in the real part is $n\frac{m}{m'}$, when the number of zeros in the real part of bitmap is z. According to [17], the expected number of items recorded in the bitmap is $-m \ln \frac{z}{m}$, under the assumption that n and m are sufficiently large and n/m is close to an arbitrary constant. In this paper, n and m satisfy this assumption as the number of distinct items n and the number of bits m are usually very large and n/m is a constant by (1). According to Alg. 1, a sampling period ends when $z = m'p$. At that time, the expected number of items recoded in the bitmap should not be less than $n\frac{m}{m'}$. Therefore, we have $n\frac{m}{m'} \leq -m \ln \frac{m'p}{m} \Rightarrow \ln \frac{m'p}{m} \leq -\frac{n}{m'} \Rightarrow m \geq m'pe^{\frac{n}{m'}}$. The minimum value of m is achieved when $m = m'pe^{\frac{n}{m'}}$. Taking the first-order derivative on the right side, we have $\frac{\mathrm{d}m}{\mathrm{d}m'} = \frac{\mathrm{d}m'pe^{\frac{n}{m'}}}{\mathrm{d}m'} = pe^{\frac{n}{m'}} - \frac{np}{m'}e^{\frac{n}{m'}} = e^{\frac{n}{m'}}p(1 - \frac{n}{m'})$

Setting $\frac{\mathrm{d}m}{\mathrm{d}m'} = 0$, we have $m' = n$. Besides, when $m' < n$, $\frac{\mathrm{d}m}{\mathrm{d}m'} < 0$; when $m' > n$, $\frac{\mathrm{d}m}{\mathrm{d}m'} > 0$. Therefore, the minimum value of m, is npe which achieves when $m' = n$. However, since $m \leq m'$, this parameter setting is valid only when $p \leq \frac{1}{e}$. For $p > \frac{1}{e}$, we always have $m' > n$ from (1) and $\frac{\mathrm{d}m}{\mathrm{d}m'} > 0$, which means the optimal setting is $m' = m$. Under this condition, we have $m' = m = m'pe^{\frac{n}{m'}} \Rightarrow 1 = pe^{\frac{n}{m'}} \Rightarrow m' = -n/\ln p$ In this case, we have $m' = -\frac{n}{\ln p}$. ■

Let M_v be the minimum number of bits required by VF to perform non-duplicate sampling. Note that M_v is the size of the real part of the virtual bitmap used by VF. Let M_t be the minimum number of bits required by TP, which is the size of the bitmap used in TP. The following corollary shows that M_v is upper-bounded by M_t. More detailed analysis shows that M_v is much smaller than M_t when the sampling probability p is small.

Corollary 1: VF requires no more memory than TP, i.e., $M_v \leq M_t$.

Proof: According to [30], the minimum size of the bitmap for the two-phase protocol is $M_t = -n/\ln p$, for any non-duplicate sampling probability p. From Theorem 2, the minimum size of the real part in the bitmap of VF is

$$M_v = \begin{cases} npe, & p < \frac{1}{e} \\ -\frac{n}{\ln p}, & \frac{1}{e} \leq p < 1 \end{cases} \tag{2}$$

Comparing M_t and M_v, we have $M_v \leq M_t$. ■

Note that when $p \geq \frac{1}{e}$, VF takes the same memory as TP does, i.e., $M_v = M_t$. When $p < \frac{1}{e}$, let's consider their ratio $\alpha = M_v/M_t = \frac{npe}{-\frac{n}{\ln p}} = -pe \ln p$. By computing its first-order derivative with respect to p, we have $\frac{\mathrm{d}\alpha}{\mathrm{d}p} = -e(\ln p + 1) > 0$.

When $p = \frac{1}{e}$, the derivative is zero and α reaches its maximum value 1. When $p < \frac{1}{e}$, the derivative is positive and we must have $\alpha < 1$, i.e., $M_v < M_t$. We plot α with respect to p in Figure 3, which suggests that VF consumes much less bits than TP, especially when p is small. For example, when $p = 0.01$, $\alpha = 0.12$, which means that TP's memory requirement is 8.3 times VF's requirement for the same data stream. (The value of β in the figure is related to the length of the sampling period, which will be discussed shortly.)

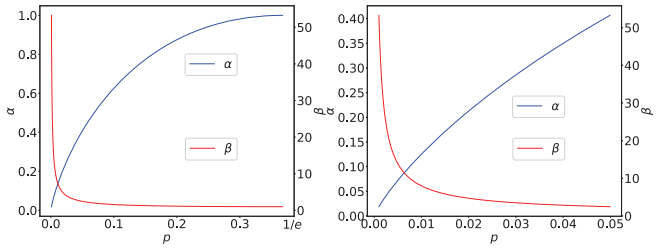
Theorem 3: Given a non-duplicate sampling probability p and a memory allocation of m bits, the expected number of distinct data items that can be recorded in VF before starting the next sampling period is

$$N_v = \begin{cases} \frac{m}{pe}, & p < \frac{1}{e} \\ -m \ln p, & \frac{1}{e} \leq p < 1 \end{cases} \tag{3}$$

The proof is trivial and thus omitted. It can be derived easily from (1).

Corollary 2: Let p be the non-duplicate sampling probability, m be the size of memory allocation, N_v be the expected number of distinct data items that can be recorded by VF in a sampling period, and N_t be the expected number of distinct items that can be recorded by TP in a sampling period. It holds true that $N_v \geq N_t$.

Proof: For the two-phase protocol [30], $N_t = -m \ln p$. Comparing it with (3), we can find that for $p \geq \frac{1}{e}$, $N_v = N_t$. As for $p < \frac{1}{e}$, let $\beta = \frac{N_v}{N_t}$, we have $\beta = -\frac{1}{ep \ln p}$. Computing



(a) $p \in (0, 1/e]$ (b) $p \in (0, 0.05]$

Fig. 3: Value of α and β w.r.t. p in different ranges.

its first-order derivative of p, we have $\frac{\mathrm{d}\beta}{\mathrm{d}p} = \frac{\ln p + 1}{ep^2 \ln^2 p}$. When $p < \frac{1}{e}$, $\frac{\mathrm{d}\beta}{\mathrm{d}p} < 0$, and when $p = \frac{1}{e}$, $\beta = 1$. Therefore, when $p < \frac{1}{e}$, we have $\frac{N_v}{N_t} = \beta > 1$, which means $N_v > N_t$. ■

We plot β with respect to p in Figure 3. It shows that N_v is larger than N_t, especially when p is very small. For example, when $p = 0.01$, $\beta = 0.065$, which means the number N_v of distinct items that VF can sample before resetting for the next period is 15.4 times that of TP, given the same amount of memory allocation.

D. Sampling Probability Adaptation

How do we determine the value of the sampling probability p? That will be application-dependent. We provide a mechanism and explain it through a network example as shown in Figure 1, where an arrival packet stream is sampled to avoid overrunning the processing capacity of the spread measurement module, where a data item is extracted from each packet — the item may be a packet header field, a combination of several fields or even data from packet payload. Suppose we set an initial sampling probability empirically. Due to traffic dynamics, the rate of sampled items that go into the module may evolve over time as the arrival packet rate changes.

First, consider the case where the sampling probability becomes too high. The consequence is that too many items are sampled, beyond what the measurement module can process in time. To deal with transient overloading, we place sampled items in a queue, which will be reduced or even emptied when the overloading condition eases. For persistent overloading, however, the queue length will keep increasing. When it passes a threshold, we will need to reduce the sampling probability to prevent overflow of the queue.

The design of VF can be modified to support real-time decrease of sampling probability by always setting m and m' to the powers of 2. With Theorem 2, we choose the expected sampling period n to a power of 2, which makes m a power of 2, assuming $p < \frac{1}{e}$. Even for $p \geq \frac{1}{e}$, we can round m down to the closest power of 2. Similarly, we round m' up to the closest power of 2. Such sub-optimal values of m and m' can in fact support a larger period than n. While it requires more bits (m) than the minimum specified in Theorem 2, it can now support dynamic decrease of sampling probability p as follows: Suppose $m' = 2^{l_1}$ and $m = 2^{l_2}$ with $l_1 \geq l_2$. If we want to reduce the sampling probability from p to $\frac{p}{2}$, we simply change $m' = 2^{l_1+1}$ and change the condition in Step 3 to $h(x) < \frac{mm'p}{2z}$. The probabilities of passing Steps 1, 2 and 3 are $\frac{m}{m'}$, $\frac{z}{m}$ and $\frac{\frac{mm'p}{2z}}{m}$, respectively, and their product is $\frac{p}{2}$, which is sampling probability of the whole filter. The reason for m' and m to be powers of 2 is to ensure duplicate filtering upon change in the sampling probability: Suppose that item x first appears before decrease of the sampling probability. Its bit index, $h(x) = H(x) \mod m'$, is simply the last l_1 bits of $H(x)$. There are two cases:

1) If $h(x)$ is in the virtual part, $h(x) > m$ and item x is filtered. Now consider a subsequent appearance of x after decrease of the sampling probability, i.e., l_1 is increased by one, which adds a leading bit to $h(x)$. Thus, we still have $h(x) > m$. Item x remains filtered.
2) If $h(x)$ is in the real part, its first $l_1 - l_2$ bits must be zeros. For example, $h(x) = 0001011$ with $l_1 = 7$ and $l_2 = 4$. Now consider a subsequent appearance of x after decrease of the sampling probability, i.e., l_1 is increased by one. The increased bit may be 0 or 1. Following the above example, $h(x)$ is now 0001011 or 1001011. If $h(x)$ is 0001011, the index is in the real part and we know that $B[1011]$ is already set to one earlier due to the first appearance of x. If $h(x)$ is 1001011, the index is in the virtual part. In both cases, this subsequent appearance of x will be filtered out.

If cutting p by half does not stop the growth of the queue, the above process of reducing the sampling probability is repeated.

Second, consider the case where the sampling probability becomes too small, which is signalled when the queue to the measurement module remains empty. Unlike the previous case of queue overflow (which needs to be handled immediately), lower sampling rate does not cause any correctness problem but may affect the measurement accuracy. VF may wait until the next sampling period to increase the sampling probability.

In practice, for quick convergence to an appropriate sampling probability, we can begin with a relatively high value and decrease it in real time towards a value that does not overrun the measurement module. After that, VF will adapt the sampling probability to the arrival traffic dynamics based on the methods described above.

E. Non-duplicate Distribution Sampling

We consider a new problem, *non-duplicate distribution sampling*, which is defined as follows: Given a series of k probabilities, p_i, $1 \leq i \leq k$, with $\sum_{i \in [1,k]} p_i \leq 1$, for the next received item x, if it is the first time that x appears in the stream, we let x pass the filter and output index i with probability p_i, $1 \leq i \leq k$; if it is not the first time, we block x. For example, if $k = 2$, $p_1 = 10\%$ and $p_2 = 20\%$, at the first appearance of item x, it has a probability of 10% to pass the filter with index 1, a probability of 20% to pass the filter with index 2, and a probability of 70% to be blocked. The non-duplicate distribution problem has not been studied before. Below we first give several applications for non-duplicate distribution sampling.

Consider a traffic measurement system where multiple measurement tasks are implemented. Some of them, such

as super spreader detection [21], [32], [25], [33], only care about flows with large spreads, and we can use low sampling probabilities to feed into these tasks so as to reduce overhead. Other measurement tasks may need information about flows of medium or small spreads for broader-scoped studies, and we use larger sampling probabilities. Instead of repeating the sampling operation for each task, we can perform it once, with the same overhead of a single sampling operation, to implement non-duplicate distribution sampling, which ensures that each item is selected at most once among all tasks with their designated probabilities, as long as the sum of all sampling probabilities does not exceed one.

Even for a single task that requires non-duplicate items, if it is executed on a multi-core processor, we can still perform the above non-duplicate distribution sampling to filter out duplicates and select items for different cores to process, with the same sampling probabilities for equal loads. Still consider a multi-core processor. If there are multiple different tasks, each assigned to a different core, then the sampling probabilities will be different, depending on the relative complexities of the tasks and their sampling requirements, as explained in the previous paragraph.

We now extend the virtual filter algorithm to support non-duplicate distribution sampling. Let m be the size of allocated memory and $p^* = \sum_{1\le i\le k} p_i$. As usual, we set m' to $\frac{m}{p^* e}$ if $p* \le \frac{1}{e}$, and to $-m \ln p^*$ otherwise. Steps 1 and 2 of the VF algorithm remain the same. Step 3 changes as follows.

Step 3: Let $p_0 = 0$. We check whether $\exists i \in [1, k]$, $\frac{mm' \sum_{0\le j\le i-1} p_j}{z} \le h(x) < \frac{mm' \sum_{0\le j\le i} p_j}{z}$. If so, we pass x through with index i as output; otherwise, we block it.

Theorem 4: Any data item will have a probability of p_i to pass the filter with index i at its first appearance, where $1 \le i \le k$. It is blocked for further appearances. No data item will pass the filter more than once.

Proof: The probability of an item x passing the first two steps for its first appearance is $\frac{z}{m'}$, where z is number of zero bits in real part of bitmap in VF. Besides, we know $h(x) < m$. Therefore, probability of x being sampled with index i is

$$\frac{z}{m'} \times \frac{\frac{mm'(\sum_{0\le j\le i} p_j - \sum_{0\le j\le i-1} p_j)}{z}}{m} = \frac{z}{m'} \times \frac{\frac{mm' p_i}{z}}{m} = p_i \quad (4)$$

For the further appearances, it will be blocked by second step since $B[h(x)]$ has been set to 1. The range of $h(x)$ for index i is $(\frac{mm' \sum_{0\le j\le i-1} p_j}{z}, \frac{mm' \sum_{0\le j\le i} p_j}{z})$ which does not have overlap with other ranges. Therefore, if an item x passed the first two steps, it can only be sampled with one index. ■

Our experimental results will confirm that, through non-duplicate distribution sampling, the sampled data items are split into k output streams, each having a different index, with a sampling rate of p_i, $1 \le i \le k$.

F. Spread Measurement with Non-duplicate Sampling

We now apply non-duplicate sampling to flow spread measurement [15], [16], which produces the spread estimates for each flow in the data stream. Note that our non-duplicate sampling can remove duplicates while sampling on distinct items with the same probability p. It can actually turn spread measurement into size measurement. The size of a flow f after our non-duplicate sampling is approximately $p * s_f$, where s_f is spread of f. Therefore, we can easily estimate flow spread s_f by measuring flow size after non-duplicate sampling using any algorithm for flow size measurement. In experiments, we choose Counter-Min with Conservative Update (CU) [12] as the algorithm for flow size measurement because it is accurate and memory/time efficient.

The data structure in CU is d counter arrays of length l, denoted as $C_i, 0 \le i < d$. To record a sampled packet of flow f, we use d independent hash functions $H_i(\cdot)$ to map it to d counters $C_i[H_i(f)]$ and increase the counters with the minimum value by 1. The query operation produces spread estimate of f as $\hat{s}_f = \min\{C_i[H_i(f)], 0 \le i < d\}/p$.

Most existing algorithms, e.g, vHLL [34] and vSketch [16], for flow spread measurement [16], [15] rely on specially designed data structures like HLL [19] to remove duplicates, which require more computations when recording and querying. By comparison, CU only needs d hash functions for recording and querying. We compare the performance of VF combined with CU with existing flow spread measurement solutions in Section IV.

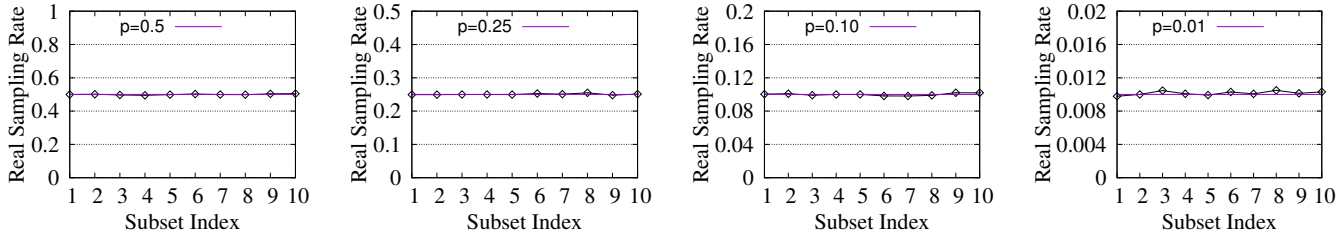
IV. Experimental Evaluation

We evaluate the performance of the proposed VF through experiments based on real-word data traces. We also compare VF with the only non-duplicate sampling work, TP. In addition, we perform two application case studies on flow spread estimation and super spreader detection, respectively, in comparison with the best prior art.

A. Experimental Setting

We have implemented (1) the proposed VF; (2) the only prior work on non-duplicate sampling, TP [30]; (3) the state-of-the-art prior work that performs flow spread estimation, vHLL [34] and vSkt(HLL) [16]; and (4) the state-of-the-art prior work that performs super spreader detection, SpreadSketch (SS) [25]. vHLL and vSkt use HyperLogLog registers [35] and SS uses multi-resolution bitmaps [36], [37]. VF under different sampling probability p is denoted as VF(p). The experiments are performed on a computer with Inter Core Xeon W-2135 3.7GHz and 32 GB memory.

The data traces used in our evaluation are real Internet traffic traces downloaded from CAIDA [38]. We use 10 traces, each containing around 20M packets. Each experiment is performed over 10 traces and we present the average results. Flow label is defined as destination address and element is the source address, both carried in each packet's header, which has the application of DDoS detection. Flow spread is the number of distinct sources that communicate with a destination. Packets will be distinct if they possess different flow labels or elements. Each trace contains around 430k distinct packets, i.e., $n \approx$430k.



(a) p=0.5 (b) p=0.25 (c) p=0.1 (d) p=0.01

Fig. 4: Actual sampling rate w.r.t. subset index, under different given sampling probability p. The difference between actual sampling rate and p is within $0.02p$, when $p \geq 0.1$, and within $0.05p$ when $p = 0.01$.

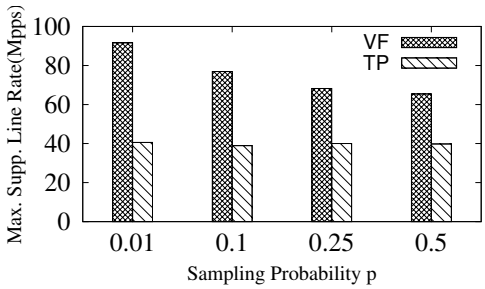


Fig. 5: Maximum supported line rate of VF in comparison with TP, under different sampling probabilities p. VF improves maximum supported line rate by over 64%, compared to TP.

The parameters of VF, i.e., m' and m are set when n and p given. n can be obtained from the real traffic traces and p will be given in each specific figure. We stress that m is the size of the real part of bitmap in VF, while m' is size of the whole bitmap in VF, including the virtual part. Therefore, we use m to denote the memory allocation of VF. We follow the parameter settings of TP, vHLL and vSkt(HLL) in the original papers. Specifically, the bitmap length of TP can be obtained according to the equations in the Performance Analysis Section of [30] when n and p are given. The HLL register is 5bits and each flow is mapped to 128 registers for vHLL and vSkt(HLL).

The evaluation is separated into four categories. The first compares the sampling performance of VF with TP. The second evaluates the performance of sampling probability adaptation. The third demonstrates the performance of non-duplicate distribution sampling of VF. The fourth compares the performance of VF for flow spread measurement with the state of the art.

B. Sampling Performance

To the best of our knowledge, TP is the only prior work that can do non-duplicate sampling, which cannot be done by traditional sampling methods. Therefore, we evaluate the sampling performance of VF in comparison with TP. The metrics are listed as follows.

- Actual sampling rate. We compare the actual sampling rate with the given sampling probability p.
- Maximum supported line rate. We measure the maximum line rate the non-duplicate sampling can catch up when processing packet streams. The unit is million packets per second, abbreviated as Mpps. Mpps is changed to Gbps if multiplying the average packet size (kbits) in the trace — if the average packet size is 1kbits, 1Mpps and 1Gbps represent the same maximum supported line rate.
- Memory requirement. It is defined as the least memory VF/TP need in order to support non-duplicate sampling of p on traffic trace with n distinct packets.
- Maximum supported n. It is defined as the maximum number of distinct packets VF/TP can support under given sampling probability p and memory allocation m. Large maximum n means longer sampling period.

p	0.5	0.25	0.1	0.05	0.01	0.005
m(VF)/m(TP)	1	0.94	0.64	0.40	0.12	0.07

TABLE I: Ratio of memory requirement of VF m(VF) over that of TP m (TP) under different p, regardless of n.

Actual sampling rate: We evaluate the actual sampling rate of ten random chosen subsets, each containing around 5k distinct packets, under given sampling probability p= 0.01, 0.1, 0.25 and 0.5, respectively. The results in Figure 4 show that the actual sampling rate is close to p, especially when p is large.

Maximum supported line rate: We compare VF with TP and plot the results in Figure 5. VF achieves higher maximum line rate, especially when p is small, e.g. 0.01. Compared to TP, VF improves the maximum line rate by 64%-125% when p decreases from 0.5 to 0.01.

Memory requirement: Table I shows the ratios of memory requirement of VF over that of TP under different p. We stress that the ratios are not affected by n. When $p = 0.5$ ($\geq 1/e$), VF and TP need the same amount of memory. When p decreases, the ratio decreases. For instance, when p=0.01, VF only needs 12% of the memory that TP needs.

Maximum supported n: Table II shows the maximum n VF and TP can support under given sampling probability p and m. As we can see, VF can support sampling data streams with larger n. For instance, when $p = 0.01$ and $m = 1$Mbits, maximum n for VF is 36.78M and for TP is 4.60M. In practical scenarios, larger n means longer sampling period.

C. Performance of Sampling Probability Adaptation

Recall that VF supports sampling probability adaptation, which is described in Section III-D. Specifically, VF can decrease the sampling probability by half immediately, while maintaining the non-duplicate sampling function. To evaluate the sampling performance when the sampling probability is cutting by half, we adopt the actual sampling rate as the metric.

m(Mbits)	0.1		0.5		1		5		10		50	
Alg. / p	VF	TP	VF	TP	VF	TP	VF	TP	VF	TP	VF	TP
0.5	0.07	0.07	0.35	0.35	0.69	0.69	3.47	3.47	6.93	6.93	34.66	34.66
0.25	0.15	0.14	0.74	0.69	1.47	1.39	7.36	6.93	14.72	13.86	73.58	69.31
0.10	0.37	0.23	1.84	1.15	3.68	2.30	18.39	11.51	36.79	23.03	183.94	115.13
0.05	0.74	0.30	3.68	1.50	7.36	3.00	36.79	14.98	73.58	29.96	367.88	149.79
0.01	3.67	0.46	18.39	2.30	36.78	4.60	183.93	23.02	367.87	46.05	1839.39	230.02
0.005	7.36	0.53	36.79	2.65	73.58	5.30	367.88	26.49	735.76	52.98	3678.79	264.92

TABLE II: Maximum supported $n(\times 10^6)$ of VF and TP under different sampling probability p, and memory allocation m. VF supports much larger n compared to TP, especially when p is small.

Round	Round 0		Round 1		Round 2		Round 3	
Initial p	p	$\hat{p}$	p	$\hat{p}$	p	$\hat{p}$	p	$\hat{p}$
0.4	0.40	0.4005	0.20	0.2018	0.1	0.0997	0.05	0.0500
0.5	0.50	0.5002	0.25	0.2517	0.125	0.1242	0.0625	0.0624
0.6	0.60	0.5981	0.30	0.3015	0.15	0.1496	0.075	0.0752
0.7	0.70	0.6978	0.35	0.3501	0.175	0.1746	0.0875	0.0879

TABLE III: Actual sampling rate $\hat{p}$ v.s. given sampling probability p for each round of cutting the sampling probability by half under different initial sampling probabilities. $\hat{p}$ in each round is very close to p for each initial sampling probability.

In the experiment, every time the number of distinct processed packets reaches a certain amount, i.e, 100k, VF adjusts the sampling probability by half. The initial sampling probability has been cut by half for three times. The measurement period can be segmented to four rounds by the value of sampling probability, i.e., round 0, round 1, round 2 and round 3. We list the actual sampling probability of each round for VF under different initial sampling probabilities in Table III. As we can see, VF can do sampling precisely for each round regardless of the initial sampling probability.

D. Performance of Non-duplicate Distribution Sampling

To investigate the sampling performance, we adopt the actual sampling rate and maximum supported line rate as the metrics. The number of probabilities k is set as 5. In practice, each probability may vary and follow different distributions. Here, we consider two distributions, even distribution and geometric distribution. Under even distribution, each probability is p^*/k. Under geometric distribution, The probability for each index is $\{\frac{p^*}{2^1}, \frac{p^*}{2^2}, ..., \frac{p^*}{2^{k-2}}, \frac{p^*}{2^{k-1}}, \frac{p^*}{2^{k-1}}\}$, respectively. We list the actual sampling probability of each index for VF under different p^* in Table IV. As we can see, VF can do sampling precisely for each index of probability p_i.

We also evaluate how non-duplicate distribution sampling can help to reduce the processing overhead. Without non-duplicate distribution sampling, k VFs are required for non-duplicate sampling operations for k probabilities. In contrast, with non-duplicate distribution sampling, a single VF can do sampling operations for k probabilities. We list the maximum supported line rates of these two approaches in Table V, which shows that with non-duplicate distribution sampling, the maximum supported line rate will be approximately $3-4$ times that when sampling operations are executed separately. The reason is that k separate VFs require k hash operations to process a packet while a single VF for non-duplicate distribution sampling only need one.

E. Case Study: Flow Spread Estimation

We now expand the evaluation to a case study of flow spread measurement. Since VF can remove duplicates, we only need a sketch to store the flow size, which is simpler compared to traditional sketches for spread measurement. Here, we use the classical and accurate one, i.e, CU [12]. Without loss of generality, we also abbreviate our method as VF or VF(p) under a specific sampling probability p. The spread produced by VF is the size stored in CU divided over p. Although TP also does non-duplicate sampling, it uses a hashmap to store the flow key and its size without memory limitation. Therefore, we only compare VF with the best sketches for flow spread measurement, i.e., vHLL [15] and vSkt(HLL) [16]. We adopt the number of arrays d=3 for CU. For VF itself, we take p=0.01,0.1,0.25,0.5 as representative sampling probabilities. For fair comparison all the sketch data structures are allocated the same memory. We use three metrics for evaluation.

- Absolute error. The average absolute error is defined as $\sum |s_f - \hat{s}_f|/N$, where $\hat{s}_f$ and s_f are the estimated and actual spread of flow f, respectively, and N is the number of flows in the flow set.
- Relative error. The average relative error is defined as $\sum \frac{|s_f - \hat{s}_f|}{N s_f}$.
- Maximum supported line rate. It has been defined before.

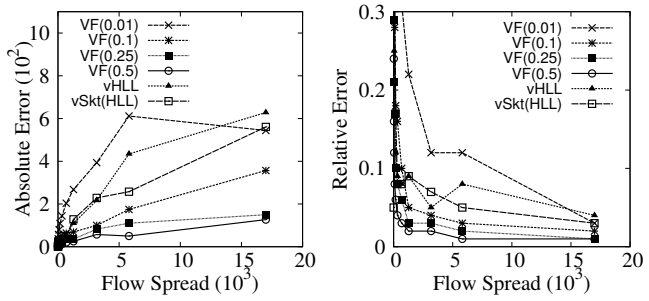
Estimation Accuracy: We first present the absolute error and relative error of all algorithms under 2Mb memory, shown in Figures 6(a) and 6(b), respectively. The flows are placed in bins based on their actual spreads (which can be found directly from the traffic traces). The spread bins are $[2^i, 2^{i+1}), i \geq 0$. We average the absolute/relative error of flows in each bin and plot a point in the figure. The results show that the accuracy of VF(p) improves as we increase p due to smaller sampling error. The average absolute error of VF(p) is much smaller than that of vHLL or vSkt(HLL) when $p = 0.1, 0.25, 0.5$. For example, VF(0.5) reduces average relative error by up to 80% and 78%, respectively, compared to vHLL and vSkt(HLL). The performances of vHLL and vSkt(HLL) are generally comparable for small/medium flows. Similar conclusions can

p^*	0.5				0.25				0.1			
Distrib.	Even		Geo.		Even		Geo.		Even		Geo.	
i	p	$\hat{p}$	p	$\hat{p}$	p	$\hat{p}$	p	$\hat{p}$	p	$\hat{p}$	p	$\hat{p}$
1	0.10	0.1007	0.25	0.24948	0.05	0.04942	0.125	0.12462	0.02	0.01982	0.05000	0.04950
2	0.10	0.09963	0.125	0.12522	0.05	0.04979	0.0625	0.06238	0.02	0.01991	0.02500	0.02515
3	0.10	0.09999	0.0625	0.06262	0.05	0.05046	0.03125	0.03134	0.02	0.01991	0.01250	0.01230
4	0.10	0.01033	0.03125	0.03132	0.05	0.04970	0.01563	0.01552	0.02	0.01995	0.00625	0.00618
5	0.10	0.01001	0.03125	0.03142	0.05	0.04985	0.01563	0.01535	0.02	0.01983	0.00625	0.00626

TABLE IV: Actual sampling rate $\hat{p}$ v.s. given sampling probability p for each index under different distributions, p^* and k=5. The actual sampling rate for each index i is very close to the given sampling probability for each index.

p^*	0.5		0.25		0.1	
Dist. Type	Even	Geo.	Even	Geo.	Even	Geo.
Separate Sampling	15.4	15.3	16.3	15.9	18.3	17.9
Dist. Sampling	48.3	47.1	50.1	49.5	56.1	54.9

TABLE V: Maximum supported line rate (Mpps) comparison of VF for a given sampling probability distribution under two different approaches. One is that sampling operations are executed using k separate VFs while the other is that sampling operations are executed by non-duplicate distribution sampling using a single VF.



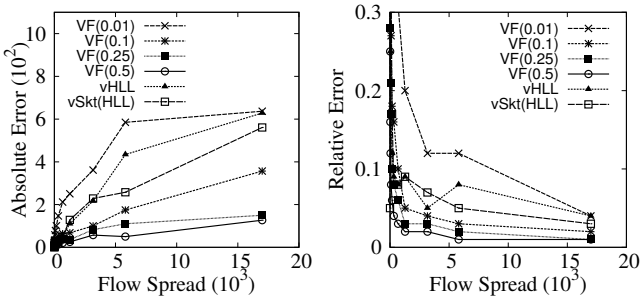
(a) absolute errors (b) relative errors

Fig. 6: Estimation accuracy of VF(p), vHLL, and vSkt(HLL) when all algorithms are allocated 2Mb memory. The algorithm of VF(0.5) reduces average relative error by up to 80% and 78%, respectively, compared to vHLL and vSkt(HLL). When p is very small, the error of VF(p) becomes worse due to its increasing sampling error.

be drawn from Figure 6(b), which compares all the algorithms in terms of the average relative errors.

We evaluate the impact of memory allocation on estimation accuracy of VF, by cutting the memory of VF by half (1Mb) and keeping the memory of vHLL and vSkt(HLL) as 2Mb. The results in Figure 7 show that even if VF is allocated 1Mb memory, its measurement accuracy is very similar to that under 2Mb. Therefore, similar conclusion can be drawn here as well. The reason is that VF transforms flow spread measurement to flow size measurement, which can be well handled by CU even in a tight memory.

Maximum supported line rate: We compare maximum supported line rates of VF(0.01), VF(0.1), VF(0.25), VF(0.5), CU, vHLL and vSkt(HLL) in Figure 8. CU can only measure flow size. It cannot measure flow spread. VF, vHLL and vSkt(HLL) can measure flow spread. We stress that our VF for flow spread measurement uses CU to process the sampled packets. The results reveal two points. The first point is that flow size measurement is simpler than flow spread measurement. Specifically, the maximum supported line rate of CU is much larger



(a) absolute errors (b) relative errors

Fig. 7: Estimation accuracy of VF(p), vHLL, and vSkt(HLL), when cutting the memory of VF by half and keeping the memories of vHLL and vSkt(HLL) as 2Mb memory. The error of VF under 1 Mb is very close to that under 2Mb.

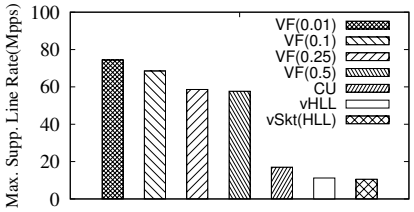


Fig. 8: Maximum supported line rate of VF(p) in comparison with CU, vHLL and vSkt(HLL). The maximum supported line rate of CU is 50% larger than vHLL and vSkt(HLL), and VF can improve the maximum supported line rate by over 2.29 times on the basis of CU.

than vHLL and vSkt(HLL), which enhances our motivation that we use VF to turn flow spread measurement to flow size measurement by removing duplicate. The second is that with the help of non-duplicate sampling done by VF, the maximum supported line rate can be much improved further.

V. Conclusion

This paper proposes a new *virtual filter algorithm* that supports non-duplicate sampling, which is substantially different from traditional sampling. It increases throughput by around 100% and reduces memory requirement by more than one magnitude when comparing with the only prior non-duplicate sampling work, especially when the sampling probability is small. We extend the basic algorithm for non-duplicate distribution sampling and flow spread measurement. When compared with the state-of-the-art, we find that our algorithm can perform better (higher accuracy, higher throughput) even when the memory is much tighter.

Acknowledgments

This work is funded by NSF grant CNS-1719222.

REFERENCES

[1] T. Benson, A. Anand, A. Akella, and M. Zhang, "MicroTE: Fine Grained Traffic Engineering for Data Centers," in *Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies*, 2011, pp. 1–12.

[2] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford, and F. True, "Deriving Traffic Demands for Operational IP Networks: Methodology and Experience," *IEEE/ACM Transactions On Networking*, vol. 9, no. 3, pp. 265–279, 2001.

[3] Y. Xie, V. Sekar, D. A. Maltz, M. K. Reiter, and H. Zhang, "Worm Origin Identification using Random Moonwalks," in *2005 IEEE Symposium on Security and Privacy (S&P'05)*. IEEE, 2005, pp. 242–256.

[4] Cisco, "Cisco IOS NetFlow," Online. [Online]. Available: http://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html

[5] Inmon Corporation, "sFlow Accuracy and Billing," Online. [Online]. Available: https://inmon.com/technology/

[6] Z. Liu, R. Ben-Basat, G. Einziger, Y. Kassner, V. Braverman, R. Friedman, and V. Sekar, "Nitrosketch: Robust and general sketch-based monitoring in software switches," in *Proceedings of the ACM Special Interest Group on Data Communication*, 2019, pp. 334–350.

[7] T. Yang, J. Jiang, P. Liu, Q. Huang, J. Gong, Y. Zhou, R. Miao, X. Li, and S. Uhlig, "Elastic Sketch: Adaptive and Fast Network-wide Measurements," *Proc. of ACM SIGCOMM*, August 2018.

[8] Q. Huang, P. P. C. Lee, and Y. Bao, "SketchLearn: Relieving User Burdens in Approximate Measurement with Automated Statistical Inference," *Proc. of ACM SIGCOMM*, pp. 576 – 590, August 2018.

[9] Q. Huang, X. Jin, P. P. C. Lee, R. Li, L. Tang, Y. Chen, and G. Zhang, "SketchVisor: Robust Network Measurement for Software Packet Processing," *Proc. of ACM SIGCOMM*, 2017.

[10] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman†, "One Sketch to Rule Them All: Rethinking Network Flow Monitoring with UnivMon," *Proc. of ACM Sigcomm*, 2016.

[11] G. Cormode and S. Muthukrishnan, "An Improved Data Stream Summary: the Count-Min Sketch and Its Applications," *Proc. of LATIN*, 2004.

[12] C. Estan and G. Varghese, "New Directions in Traffic Measurement and Accounting," *Proc. of ACM SIGCOMM*, August 2002.

[13] M. Charikar, K. Chen, and M. Farach-Colton, "Finding Frequent Items in Data Streams," *Proc. of International Colloquium on Automata, Languages, and Programming (ICALP)*, July 2002.

[14] C. Ma, H. Wang, O. Odegbile, and S. Chen, "Noise Measurement and Removal for Data Streaming Algorithms with Network Applications," in *2021 IFIP Networking Conference (IFIP Networking)*. IEEE, 2021, pp. 1–9.

[15] Q. Xiao, S. Chen, M. Chen, and Y. Ying, "Hyper-Compact Virtual Estimators for Big Network Data Based on Register Sharing," *in Proc. of ACM SIGMETRICS*, 2015.

[16] 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, Dec. 2019.

[17] 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, 1990.

[18] 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.

[19] 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.

[20] 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, pp. 1040–1052, 2021.

[21] G. Cormode and S. Muthukrishnan, "Space Efficient Mining of Multigraph Streams," *Proc. of ACM PODS*, June 2005.

[22] S. Ganguly, M. Garofalakis, R. Rastogi, and K. Sabnani, "Streaming Algorithms for Robust, Real-time Detection of Ddos Attacks," in *27th International Conference on Distributed Computing Systems (ICDCS'07)*. IEEE, 2007, pp. 4–4.

[23] W. Liu, W. Qu, J. Gong, and K. Li, "Detection of superpoints using a vector bloom filter," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 3, pp. 514–527, 2015.

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

[25] 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.

[26] 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.

[27] 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.

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

[29] B. H. Bloom, "Space/Time Trade-offs in Hash Coding with Allowable Errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[30] 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.

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

[32] S. Venkataraman, D. Song, P. B. Gibbons, and A. Blum, "New Streaming Algorithms for Fast Detection of Superspreaders," *Proc. of NDSS*, 2005.

[33] Y. Liu, W. Chen, and Y. Guan, "Identifying High-cardinality Hosts from Network-wide Traffic Measurements," *IEEE Transactions on Dependable and Secure Computing*, vol. 13, no. 5, pp. 547–558, 2015.

[34] 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.

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

[36] C. Estan, G. Varghese, and M. Fisk, "Bitmap Algorithms for Counting Active Flows on High Speed Links," in *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, 2003, pp. 153–166.

[37] C. Estan, G. Varghese, and M. Fish, "Bitmap Algorithms for Counting Active Flows on High-Speed Links," *IEEE/ACM Trans. on Networking*, vol. 14, no. 5, October 2006.

[38] UCSD, "CAIDA UCSD Anonymized 2015 Internet Traces on Jan. 17," http://www.caida.org/data/passive/passive_2015_dataset.xml, 2015.