Virtual Filter for Non-Duplicate Sampling With Network Applications

Chaoyi Ma, *Graduate Student Member, IEEE*, Haibo Wang, *Graduate Student Member, IEEE*, Olufemi O. Odegbile, Shigang Chen, *Fellow, IEEE*, and Dimitrios Melissourgos

***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 modified to handle sampling for multiple independent tasks with different probabilities. We also enhance the virtual filter for flow spread measurement and super spreader detection with a large measurement period.**

***Index Terms*— Non-duplicate sampling, traffic measurement, flow spread.**

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 [2]–[4]. The widely used tools, including NetFlow [5] and sFlow [6], 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.

Manuscript received September 9, 2021; revised May 9, 2022; accepted June 9, 2022; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor R. Lo Cigno. This work was supported by NSF under Grant CSR 1909077 and Grant NeTS 1719222. A conference version of this paper has been published in IEEE ICNP 2021 [1] [DOI: 10.1109/ICNP52444.2021.9651974]. *(Chaoyi Ma and Haibo Wang are co-first authors.) (Corresponding author: Chaoyi Ma.)*

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

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

Digital Object Identifier 10.1109/TNET.2022.3182694

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 [7], Elastic Sketch [8], SketchLearn [9], SketchVisor [10], UnivMon [11] and many others [12]–[15]. 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 [16]–[21], 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 [22]–[26] or detect malicious activities [27]–[29]. 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 [30] 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 [31]. 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 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 [31], 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 do non-duplicate sampling for multiple independent tasks with different probabilities. This is practically useful as some of the tasks may be interested only in large-spread flows (small sampling probabilities), while others may be interested in broad-scoped measurement (large sampling probabilities). Instead of using one virtual filter for each task, we only utilize a single virtual filter to perform non-duplicate sampling for all tasks, which significantly improves memory efficiency and reduces processing overhead.

Third, we apply the virtual filter algorithms to two measurement tasks, flow spread measurement and super spreader detection. Observing that the practical data streams are usually skewed, we optimize our virtual filter algorithms to have a large measurement period by saving the memory consumption of large flows.

Fourth, 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. As for sampling for multiple tasks, the experimental results show a much higher throughput and a much lower memory consumption when compared to the basic approach that deploys one virtual filter for each task. We also perform case studies of using non-duplicate sampling to support flow spread measurement and super spreader detection. With a novel optimized design, the measurement period of these two applications can be extended. It greatly improves the measurement throughput and surprisingly also improves measurement accuracy even when less memory is allocated.

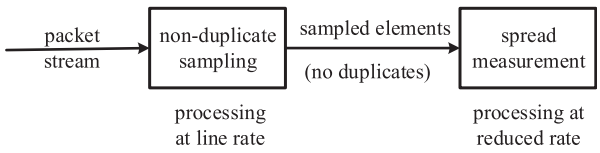


Fig. 1. System model.

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

We will apply non-duplicate sampling on network traffic measurement, as illustrated in Figure 4. 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 [28], [29]. 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 [27], [32]. 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. Related Work

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 [31]. 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. Regardless of whether the item is sampled or not in the first phase, the second phase will check 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 is sampled in the first phase and is passed in the second phase.

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. Loosely speaking for intuition only, one bit per item allows a sampling period to contain 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 $10m$ 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, $10m$ distinct items instead m items by TP in the above example.

Moreover, TP takes one hash and one random number generation 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 a 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.

A three-phase protocol [33] is extended from TP. Its objective is to improve memory efficiency of TP by adding an extra hash operation for additional sampling, which however increases its processing overhead and thus decreases the line rate that it can support. Because the primary performance objective of this work is to reduce the processing overhead to one hash per packet, we will choose TP for comparative study since it incurs less processing overhead than its extension. Besides overhead, the three-phase protocol [33] has the same limitations listed above as TP does, including lack of adaptability to traffic dynamics and no consideration of multiple sampling probabilities.

C. Other Related Work

Since non-duplicate sampling can be used for per-flow spread measurement and super-spreader detection (detect flows whose spread exceed an threshold), we briefly introduce some state-of-art works on these two problems that we used as benchmarks in the experimental evaluation.

vSkt(HLL) [17] and vHLL [16] utilized HLL [20], a single flow spread estimator to do per-flow spread measurement. A HLL estimator contains m HLL registers where each element of the flow will be randomly recorded in one of them using a uniform hash function. After recording, the spread of the flow can be estimated from the values in all registers. vHLL keeps an array of HLL registers of length l. The m

registers for a flow f are randomly selected from the array using m independent hash functions. In this way, several flows may share the same register, e.g., some elements from other flows may be recorded in f' register which causes noise. vHLL assumes that all elements except those of f are randomly mapped in l registers and thus $\frac{m}{l}$ of them are the noise in f' registers. Therefore, the noise can be calculated if we can estimate the total spread and this can be done by treating the whole array as a HLL estimator. vSkt(HLL) is slightly different from vHLL. It keeps m HLL register array and it selects one register from each array to compose the HLL estimator of a flow f.

CMH [22] and SpreadSketch [26] replace the counters in CountMin [12] with single flow spread estimators such as multi-resolution bitmap [34] for super spreader detection. They keep d arrays of estimators, each flow is mapped to d estimators, one in each array. All elements of a flow will be recorded in all d estimators of it. After recording, the estimate is the smallest value among the d estimators since it contains the minimum noise. CMH uses a heap to store the current top-q largest flows and their spread estimates. Every time it records an element of a flow, it estimates its spread and tries to update the heap. Since the estimating operation is much slower than the recording operation, CMH is not efficient. SpreadSketch improves CMH by maintaining additional values for each estimator. It generates a geometric hash value for each element and stores the flow label of the element with the largest hash value in each estimator. Here the geometric hash is a type of hash function that has an output of i with a probability of $\frac{1}{2^i}, i \geq 1$. After a measurement period, it estimates spreads of all flows whose labels are stored and reports the super spreaders.

Since the single flow spread estimators (e.g., HLL) is limited in measurement accuracy and memory consumption, the above algorithms built on top of the single flow spread estimators are also limited for per-flow spread measurement or super spreader detection. We will show these in Section VIII when comparing them with our algorithm.

IV. Non-Duplicate Sampling With Virtual Filter

We first describe the proposed virtual filter algorithm for non-duplicate sampling and then discuss how to adapt the sampling probability during item recording.

A. 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 $B[0] \dots B[m-1]$ the real part of the filter and $B[m] \dots B[m'-1]$ 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

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

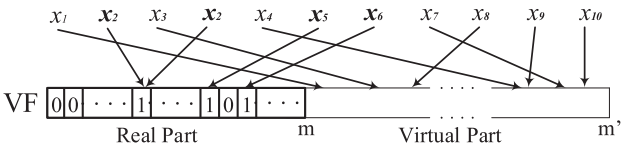


Fig. 2. Among the items of the data stream, x_2, x_5 and x_6 are hashed to the real part of the filter. Other items are hashed to the virtual part and thus blocked. The second appearance of x_2 is blocked by Step 2 because the bit it hashes to has been set to one by the first appearance of x_2. The condition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

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.

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 2. The issue of setting the sampling probability under dynamic traffic conditions will be addressed later in Section IV-B. 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 3, 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 [18], 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

$$\frac{\mathrm{d}m}{\mathrm{d}m'} = \frac{\mathrm{d}m' p e^{\frac{n}{m'}}}{\mathrm{d}m'} = p e^{\frac{n}{m'}} - \frac{np}{m'} e^{\frac{n}{m'}} = e^{\frac{n}{m'}} p(1 - \frac{n}{m'}).$$

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

$$m' = m = m' p e^{\frac{n}{m'}} \Rightarrow 1 = p e^{\frac{n}{m'}} \Rightarrow m' = -n/\ln p.$$

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 [31], 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 $M_v \leq M_t$. □

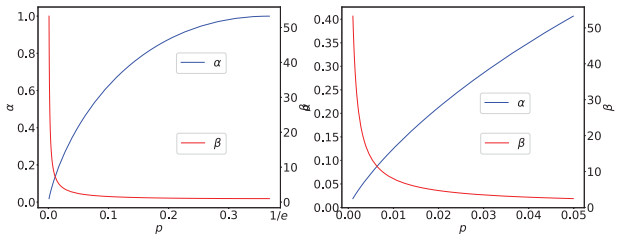
Note that when $p \geq \frac{1}{e}$, VF takes the same memory as TP does, i.e., $M_v = M_t$. When $p < \frac{1}{e}$, let's consider their ratio $\alpha = M_v/M_t = \frac{npe}{-\frac{n}{\ln p}} = -pe \ln p$. By computing its first-order derivative with respect to p, we have $\frac{\mathrm{d}\alpha}{\mathrm{d}p}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

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.)

Given the sampling probability p, the memory usage m and the expected number of distinct items to be processed in VF, i.e., n are a trade-off. Theorem 2 fixes n and explores how to minimize the usage memory. Alternatively, we provide another view on the maximum n given m by giving the following corollaries.

Corollary 2: Given a non-duplicate sampling probability p and a memory allocation of m bits, the maximum expected number of distinct data items that can be recorded in VF before starting the next sampling period is

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



(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.

where the optimal setting for m' is

$$m' = \begin{cases} \frac{m}{pe}, & p < \frac{1}{e} \\ m, & \frac{1}{e} \leq p < 1 \end{cases} \tag{4}$$

The proof is omitted as it is trivial and can be easily derived from Theorem 2.

Corollary 3: 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 [31], $N_t = -m \ln p$. 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 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.

B. 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 4, where an arrival packet stream is sampled to avoid overrunning the processing capacity of the spread measurement module. 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. [1]

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.

V. Non-Duplicate Sampling for Multiple Tasks

We consider a new scenario where we need to perform independent non-duplicate sampling for multiple tasks simultaneously with different sampling probabilities. We are given a series of k probabilities, p_i, $1 \leq i \leq k$, each for a task. Upon the arrival of any item x, if it is the first time that x appears in the stream, we let x pass the filter for index i with probability p_i, $1 \leq i \leq k$; if it is not the first time, we block x. Note that these tasks are independent and the sampling should also be independent for each task. 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 and a probability of 20% to pass the filter with index 2. One item can be sampled by any number of tasks. Below we first give an application under this new scenario.

Consider a traffic measurement system where multiple measurement tasks are implemented. Some of them, such as super spreader detection [22], [26], [35], [36], 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. In this scenario, we have multiple tasks with different sampling probabilities.

A straightforward way of doing non-duplicate sampling for multiple tasks is deploying k independent virtual filters, each for a task. This requires considerable computation and memory access overhead. Instead of repeating the whole sampling operation for each task, we can use a single virtual filter to perform the non-duplicate sampling for multiple tasks. We give the algorithm below.

Let m be the size of allocated memory and $p_1 \leq p_2 \leq \cdots \leq p_k$. We set m' to $\frac{m}{p_k e}$ if $p_k \leq \frac{1}{e}$, and to $-m \ln p_k$ otherwise. Steps 1 and 2 of the VF algorithm remain the same. Step 3 changes as follows.

Step 3: For an item which has already passed Step 1 and Step 2, we generate k random number $r_i, 1 \leq i \leq k$ with in range $[0, 1)$. If $r_i < \frac{p_i m'}{z}$, we let it pass for the ith task; otherwise, we block it for the ith task.

Theorem 3: Any data item will have a probability of p_i to pass the filter for ith task at its first appearance, where $1 \leq i \leq k$. It is blocked for further appearances.

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. Therefore, probability of x being sampled for ith task is

$$\frac{z}{m'} \times \frac{p_i m'}{z} = p_i \tag{5}$$

For the further appearances, it will be blocked by second step since $B[h(x)]$ has been set to 1. □

Our experimental results will confirm that, with the new algorithm, we can achieve non-duplicate sampling for multiple tasks with different sampling probabilities and in the meanwhile significantly reduce the processing overhead and improve memory efficiency compared to the naive solution that employing k VFs for k tasks.

Here we generate a random number for each sampling task instead of directly replacing $\frac{mm'p}{z}$ with $\frac{mm'p_i}{z}$ in Step 3 because we want to make sure that the samplings for each task are independent. If we directly replace $\frac{mm'p}{z}$ with $\frac{mm'p_i}{z}$ in Step 3, then obviously a packet sampled by the jth task will be sampled by the ith task if $p_i \leq p_j$. This make the samplings become dependent among different tasks.

[1] Setting the threshold value is an empirical task: A larger threshold means more memory requirement, but will be more capable of absorbing transient overloading; whereas a smaller threshold uses less memory, but will be more sensitive to flush crowd causing the reduction in the sampling probability.

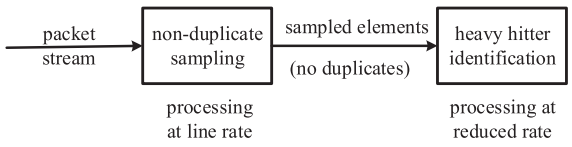


Fig. 4. Role of virtual filter: transforming complex super spreader detection to easier heavy hitter identification.

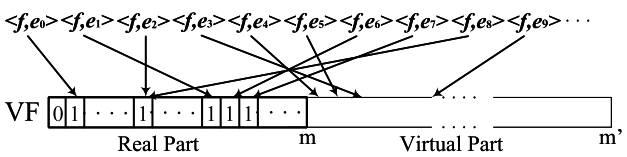


Fig. 5. A large flow f with spread s_f could occupy a large portion of bits (up to s_f) in VF.

VI. Applications on Non-Duplicate Sampling

A. Spread Measurement With Non-Duplicate Sampling

We now apply non-duplicate sampling to flow spread measurement [16], [17], 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) [13] 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 $C_i, 0 \leq i < d$. 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 $\hat{s}_f = \min\{C_i[H_i(f)], 0 \leq i < d\}/p$.

Most existing algorithms, e.g, vHLL [37] and vSketch [17], for flow spread measurement [16], [17] rely on specially designed data structures like HLL [20] 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 VIII.

B. Super Spreader Detection

Super spreaders are defined as flows whose spreads exceed a predefined threshold U_1. Due to the difficulty of spread measurement that we discussed before, the existing work on super spreader detection is limited in terms of estimation, throughput and memory efficiency. By comparison, an easier problem is *heavy hitter identification*, which is to report flows whose sizes exceed a pre-defined threshold U_2. Through non-duplicate sampling that is enabled by VF, we can apply heavy hitter identification solutions to detecting super spreaders. Specifically, we put VF in the front of a heavy hitter identification solution and set $U_2 = U_1 \cdot p$. The reported heavy hitters are super spreaders.

This paper adopts an efficient heavy hitter identification solution, CMH [22], which combines CountMin (CM) [22] with a heap of size q. It is designed for finding top-q super spreaders. To fit with threshold-based super spreader detection, we replace the heap with a hash table. Every time we record a packet in CM, we query its size. If its size exceeds U_2, the flow is a super spreader and we put the flow identifier into the hash table. Our experimental results show that our algorithm is the winner in detection accuracy and throughput when comparing with existing work. Note that when the dataset is skewed, we can also employ the new algorithm of VF in Section VII for VF to save memory.

VII. Extend Measurement Period of Virtual Filter for Spread Measurement and Super Spreader Detection

With VF, non-duplicate sampling can be achieved within a measurement period but the cross-period duplicates still exist. Given a stream, the influence of cross-period duplicates will be decreased if the measurement period increases. The ideal way is to let the whole packet stream be processed in one measurement period to remove duplicates. This goes at the cost of nearly infinity memory considering a steam with possibly infinite packets. Practically, we may relax the requirement and expect to extend the measurement period in order to minimize the influence. A straightforward way to extend the measurement period, as Theorem 2 shows, is to enlarge the memory of virtual filter. However, extra memory is not always available, especially when the measurement module as well as virtual filter is implemented on switches/routers with limited on-chip memory. This section attempts to extend the measurement period of virtual filter for spread measurement and super spreader detection with no additional memory consumption.

A large number of real-world data streams including network traffic traces are highly skewed. A small number of large flows contribute to the most total spread in the stream. Recall that under the optimal parameter setting, VF requires the number of bits that is linear to the number of distinct data items in the stream when p is fixed (see Theorem 2). For those data streams, large flows will take up a majority of the bits in VF, illustrated in Fig. 5 where we consider a packet stream and each packet is abstracted as $\langle f, e\rangle$. This section attempts to reduce the memory consumption of large flows and furthermore extend the measurement period of VF under given m and m'. To achieve this goal, we need to ensure that the number of bits that any flow can be mapped to is not larger than a pre-defined constant, n_s. The value of n_s can be adjusted depending on the resource constraints and application needs.

Our main idea is that for any flow f, we randomly select n_s bits from VF, denoted as $B[r_0(f)]$, $B[r_1(f)]$, $B[r_2(f)], \ldots, B[r_{n_s-1}(f)]$. These bits together will form an imaginary bitmap B_f, with $B_f[i] = B[r_i(f)]\ \forall 0 \leq i < n_s$; see Fig. 6. When flow f's packets are processed by VF in Step 2, they can only be recorded in bits belonging to B_f. The method is to construct an array S of n_s seeds. For any packet $\langle f, e\rangle$, before we begin Step 1 in VF, we first select a seed $S[H_*(e) \mod n_s]$ in S for element e, where $H_*(\cdot)$ is uniform hash function with a sufficient output range. We feed $\langle f, S[H_*(e) \mod n_s]\rangle$ to VF instead of $\langle f, e\rangle$.

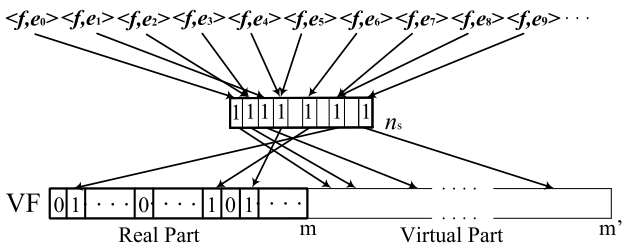


Fig. 6. Any flow f with spread s_f can occupy at most n_s bits in VF even if $s_f \geq n_s$. n_s is a pre-defined constant.

From Step 2, we know each flow f can only be mapped to the following n_s bits:

$$B[h(f, S[0])], B[h(f, S[1])], \ldots, B[h(f, S[n_s - 1])].$$

These n_s bits form the imaginary bitmap B_f for flow f mentioned in the main idea. The detailed packet recording operation is described in Alg. 2.

Next, we consider how to estimate the spread for flow f. For f's seed array S with length n_s, if there has been at least one elements in f that are hashed to $S[i]$. The bit $S[i]$ is called true bit, and false bit otherwise. Let X_f^u be the number of distinct packets required to make the number of true bits in S increase by 1 from $u-1$ to u with $1 \leq u \leq n_s$. Due to the uniformness of the hash value, X_f^u is a geometric random variable, i.e.,

$$X_f^u \sim Ge(\frac{n_s - u + 1}{n_s}) \tag{6}$$

For any packet $\langle f, e\rangle$ consider its hashed bit $S[j]$ in S. If it is a false bit, it will be blocked and not be processed further as $\langle f, S[H_*(e) \mod n_s]\rangle$ has already been processed by VF. If $S[j]$ a true bit, consider its hash value $H(f, S[j]) \mod m'$. There is a probability of m/m' to pass Step 1. In Step 2, the packet's mapped bit in B, i.e., $B[H(f, S[j]) \mod m']$ has a probability of z/m to be zero due to the uniformness of hash value $H(\cdot)$. If so the packet passes through Step 2. For Step 3, as we already know $H(f, S[j]) \mod m' < m$ as it passes through Step 1, the probability to pass throughput this step is $\frac{mm'p}{zm} = \frac{m'p}{z}$. Therefore, the packet pass through VF with a overall probability of p.

Let Y_f^v be the number of true bits in S required to make f's packets pass through the non-duplicate sampling process and eventually be recorded in CU for the vth time. Y_f^v is a geometric random variable, i.e., $Y_f^v \sim Ge(p)$.

Let V_f be the total number of times that f' packets are recorded in CU and Y be the total number of true bits in S when the recording finished. We have

$$Y = \sum_{v=1}^{V_f} Y_f^v = \frac{V_f}{p}. \tag{7}$$

Let X be the total number of distinct elements in f. We have

$$X = \sum_{u=1}^{Y} X_f^u.$$

Combining the above equation, (6) and (7), we have

$$\begin{aligned} \mathrm{E}(X) &= \mathrm{E}(\sum_{u=1}^{Y} X_f^u) = \mathrm{E}(\sum_{u=1}^{Y} \frac{n_s}{n_s - u + 1}) \\ &\approx n_s(H_{n_s} - H_{n_s - \mathrm{E}(Y)}) = n_s(H_{n_s} - H_{n_s - \frac{\mathrm{E}(V_f)}{p}}) \end{aligned}$$

H_i is the ith harmonic number with the asymptotics property of $\lim_{i\to\infty} H_i \to \ln i + \gamma$ where γ is a sufficiently small constant and can be neglected. For the special case of $\frac{\mathrm{E}(V_f)}{p} = n_s$, we have $H_0 = 0$.

CU will produce the estimate $\hat{V}_f$ for V_f by replacing the $\mathrm{E}(V_f)$ and $\mathrm{E}(X)$ by their observed values $\hat{V}_f$ and $\hat{X}$. Since $\frac{\mathrm{E}(V_f)}{p} <= n_s$ and should be integer, we set $\frac{\hat{V}_f}{p} = n_s$ if $\frac{\hat{V}_f}{p} > n_s - 1$, resulting in two cases for estimation formula.

$$\hat{X} = \begin{cases} -n_s \ln(1 - \hat{V}_f/(n_s p)), & \frac{\hat{V}_f}{p} \leq n_s - 1; \\ n_s \ln(n_s), & \text{otherwise.} \end{cases} \tag{8}$$

The above formula indicates that the maximum supported estimate ($n_s \ln(n_s)$) is related to the value of n_s. The estimation range is enlarged if we use a larger n_s, while the sampling period will be extended more if we use a smaller n_s (which we will show in the next theorem). This is a trade-off and in practice we may set $n_s \ln(n_s)$ to be slightly larger than the spread of the largest flow.

Theorem 4: Given a non-duplicate sampling probability p and a memory allocation of m bits, the maximum expected number of distinct data items that can be recorded in VF before starting the next sampling period is at least $N_v + n_L - v_L \times n_s$, where v_L is the number of flows whose spreads are larger than n_s, n_L is the number of distinct items from all flows whose spreads are larger than n_s in the first N_v data items and

$$N_v = \begin{cases} \frac{m}{pe}, & p < \frac{1}{e} \\ -m \ln p, & \frac{1}{e} \leq p < 1. \end{cases} \tag{9}$$

under the optimal setting for m' as follows:

$$m' = \begin{cases} \frac{m}{pe}, & p < \frac{1}{e} \\ m, & \frac{1}{e} \leq p < 1 \end{cases} \tag{10}$$

Proof: Given a sampling probability p and a memory allocation of m, the optimal setting of m' that can maximum the number of distinct items the filter can process is given in Theorem 2. With the optimal setting, the filter can process N_v distinct data items. In the new design, instead of $\langle f, e\rangle$, we feed $\langle f, S[H_*[e]]\rangle$ to the filter, where $H_*(e) \in [0, n_s)$. This means for any flow f, at most n_s distinct items will be feeded to the filter. After we record N_v distinct items, at most $N_v - n_L + v_L \times n_s$ items will be feeded to the filter. Therefore, we can record at least $N_v - (N_v - n_L + v_L \times n_s) = n_L - v_L \times n_s$ more distinct items. As a result, we can record at least $N_v + n_L - v_L \times n_s$ distinct item in total. □

Note that $n' \leq n$ as $n_L \geq v_L \times n_s$, which means our new design can enlarge the sampling period, e.g., process more distinct items under the same memory allocation. We will experimentally evaluate how the optimization in this subsection can enlarge the sampling period over the solution to flow spread estimation in Section VI-A in Section VIII-F.

Algorithm 2 Recording of VF for spread measurement with probability p

Input: sampling probability p, data stream
Action: perform non-duplicate sampling for spread measurement
setting m, m'
create a bitmap B of m bits, set $z = m$
create a random seeds array S of length n_s
create d counter array $C_i, 0 \leq i < d$ of length l
for each packet $\langle f, e \rangle$ **do**
  $j = H_*(e) \mod n_s$
  $i = h(f, S[j]) = H(f, S[j]) \mod m'$
  if $i < m$ **then**
    if $B[i] = 0$ **then**
      $B[i] = 1$
      if $i < \frac{mm'p}{z}$ **then**
        $m_c = \min\{C_k[H_k(f)] | 0 \leq k < d\}$
        for $k = 0$ to $d - 1$ **do**
          if $C_k[H_k(f)] = m_c$ **then**
            $C_k[H_k(f)] = C_k[H_k(f)] + 1$
      $z = z - 1$

Algorithm 3 Querying of VF for Spread Measurement With Probability p

Input: sampling probability p, flow label f, sub-flow number n_s, counter arrays $C_i, 0 \leq i < d$
Output: spread estimate of flow f
$\hat{V}_f = \min\{C_k[H_k(f)] | 0 \leq k < d\}$
$\hat{s}_f = -n_s \ln(1 - \frac{\hat{V}_f}{n_s p})$
return $\hat{s}_f$

Our new design for VF can also be applied to super spreader detection, where we want to find out flows whose spreads exceed a predefined threshold U_1. Once the a flow's spread exceeds U_1 during recording, the recording of the remaining distinct elements is not our concern and thus should be limited to extend measurement period under given memory. A simple way to solve this problem is as follows: For each arrival item $\langle f, e \rangle$, we query the spread of f and only record the item when the estimate is less than U_1, e.g., it has not been detected as a super spreader. According to Section VI-B, we need to query hash table for each item, which requires non-trivial extra computations. Employing the similar idea for extending the measurement period for spread measurement, we can easily limit the influence of large flows without extra querying operations. The main idea is to set $-n_s \ln \frac{1}{n_s} > U_1$. Since we only care about whether $s_f > U_1$, this n_s is enough to measure flows with a spread of U_1. Any extra elements will automatically be filter out since a flow can set at most n_s bits in the filter.

VIII. 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.

TABLE I

The Columns of Minimum and Maximum Give the Range of Actual Sampling Rates Measured in the Experiment, Under Different Target Sampling Probabilities p. The Difference Between Actual Sampling Rates and p Is Within $0.02p$, When $p \geq 0.1$, and Within $0.05p$ When $p = 0.01$. All Measured Results Are Rounded to Four Digits After the Decimal Point

p	minimum	maximum
$p = 0.5$	0.4988	0.5049
$p = 0.25$	0.2480	0.2547
$p = 0.1$	0.0981	0.1020
$p = 0.01$	0.0097	0.0104

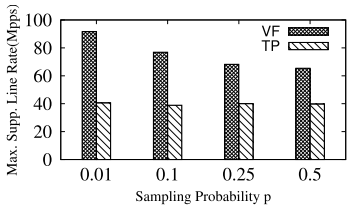


Fig. 7. Maximum supported line rate of VF in comparison with TP, under different sampling probabilities p. VF improves maximum supported line rate by over 64%, compared to TP.

A. Experimental Setting

We have implemented (1) the proposed VF; (2) the only prior work on non-duplicate sampling, TP [31]; (3) the state-of-the-art prior work that performs flow spread estimation, vHLL [37] and vSkt(HLL) [17]; and (4) the state-of-the-art prior work that performs super spreader detection, SpreadSketch (SS) [26]. vHLL and vSkt use HyperLogLog registers [38] and SS uses multi-resolution bitmaps [34], [39]. 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 set used in our evaluation are real Internet traffic traces downloaded from CAIDA [40]. We use 10 traces, each containing around 20M packets. 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., $n \approx 430$k.

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 [31] when n and p are given. The HLL

TABLE II

RATIO OF MEMORY REQUIREMENT OF VF m(VF) OVER THAT OF TP m (TP) UNDER DIFFERENT p, REGARDLESS OF n

p	0.5	0.25	0.1	0.05	0.01	0.005
m(VF)/m(TP)	1.00	0.94	0.64	0.40	0.12	0.07

register is 5bits and each flow is mapped to 128 registers for vHLL and vSkt(HLL).

The evaluation is separated into six 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 the with the state of the art. The fifth compares the performance of our optimized VF for flow spread measurement when extreme flows exist in the dataset. Finally, the sixth compares the performance of VF for super spreader detection with the best prior work.

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.

Actual sampling rate: We take 5k packets with distinct source-destination address pairs randomly from each traffic traces, then perform non-duplicate sampling under a given sampling probability p, and measure the actual sampling rate to see if it is close to p. There are ten measurements from the ten traces. Table I presents the minimum value and the maximum value of the ten actual sampling rates, which are all close to p under all different p values used in the experiment.

Maximum supported line rate: We compare VF with TP and plot the results in Figure 7. 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 and maximum supported n: Table II 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. Table III shows the ratio of the maximum n VF and TP can support under given sampling probability p. VF can support sampling data streams with larger n, especially when p is small. In practical scenarios, larger n means longer sampling period.

TABLE III

RATIO OF THE MAXIMUM SUPPORTED n OF VF AND TP UNDER DIFFERENT SAMPLING PROBABILITY p. VF SUPPORTS MUCH LARGER n COMPARED TO TP, ESPECIALLY WHEN p IS SMALL. ALL MEASURED RESULTS ARE ROUNDED TO TWO DIGITS AFTER THE DECIMAL POINT. THE RATIO REMAINS THE SAME UNDER DIFFERENT MEMORY ALLOCATIONS

p	0.5	0.25	0.1	0.05	0.01	0.005
n(VF)/n(TP)	1.00	1.06	1.59	2.45	7.88	13.88

C. Performance of Sampling Probability Adaptation

Recall that VF supports sampling probability adaptation, which is described in Section IV-B. 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. 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 IV. As we can see, VF can do sampling precisely for each round regardless of the initial sampling probability.

D. Performance of Non-Duplicate Sampling for Multiple Tasks

We evaluate the performance of the approach that uses a single VF for multiple tasks (in Section V). The baseline solution is to employ one VF for each task. To investigate the sampling performance, we adopt the actual sampling rate, the maximum supported line rate and the memory consumption as metrics. The number of probabilities k is set as 5 and the sampling probabilities are denoted as $\{p_1, p_2, p_3, p_4, p_5\}$, respectively. In practice, each probability may vary and follow different distributions. Here, we consider two distributions, linear distribution and geometric distribution. Under linear distribution, we have $p_i = \eta i$. Under geometric distribution, we have $p_i = 2^{i-1}\eta$. We list the actual sampling probability of each index for VF under different η in Table V. As we can see, VF can do sampling precisely for each index of probability p_i.

We also evaluate how using a single VF for multiple tasks can help to reduce the processing overhead. The results on the maximum supported line rates of these two approaches are listed in Table VI, which shows that using a single VF, the maximum supported line rate will be approximately 3 times that when sampling operations are executed separately. For the baseline solution, any packet will be processed by k VFs. In contrast, using a single VF for multiple tasks can only process any packet once in Step 1 and Step 2, significantly reducing the processing overhead. Table VII shows that the

TABLE IV

ACTUAL SAMPLING RATE $\hat{p}$ VS. 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. ALL MEASURED RESULTS ARE ROUNDED TO FOUR DIGITS AFTER THE DECIMAL POINT

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 V

ACTUAL SAMPLING RATE $\hat{p}$ VS. 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. ALL MEASURED RESULTS ARE ROUNDED TO FIVE DIGITS AFTER THE DECIMAL POINT

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	0.10001	0.03125	0.03132	0.02	0.01965	0.00782	0.00786	0.01	0.00983	0.00312	0.00308
2	0.20	0.20033	0.0625	0.06262	0.04	0.03970	0.01563	0.01552	0.02	0.01995	0.00625	0.00618
3	0.30	0.29999	0.125	0.12522	0.06	0.60046	0.03125	0.03134	0.03	0.02991	0.01250	0.01230
4	0.40	0.39973	0.25	0.24948	0.08	0.07949	0.0625	0.06238	0.04	0.03991	0.02500	0.02515
5	0.50	0.50070	0.50	0.04942	0.10	0.09942	0.125	0.12462	0.05	0.04982	0.05000	0.04950

TABLE VI

MAXIMUM SUPPORTED LINE RATE (MPPS) COMPARISON OF VF FOR A GIVEN SERIES OF SAMPLING PROBABILITIES 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 SAMPLING FOR MULTIPLE TASKS USING A SINGLE VF. ALL MEASURED RESULTS ARE ROUNDED TO ONE DIGIT AFTER THE DECIMAL POINT

p^*	0.5		0.25		0.1	
Dist. Type	Even	Geo.	Even	Geo.	Even	Geo.
Using k VFs	14.4	15.0	15.6	16.9	17.3	17.8
Using Single VF	43.3	42.1	45.1	44.5	46.1	46.9

TABLE VII

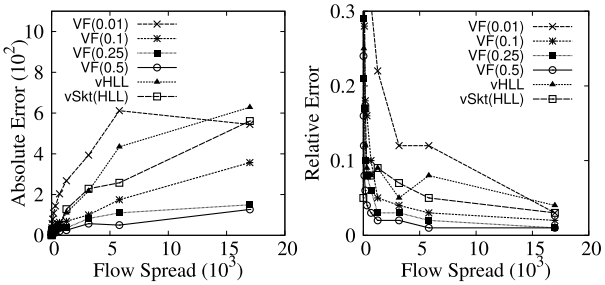
RATIO OF MEMORY REQUIREMENT OF THE APPROACH USING k VFS OVER THAT USING A SINGLE VF FOR A GIVEN SERIES OF SAMPLING PROBABILITIES. ALL MEASURED RESULTS ARE ROUNDED TO TWO DIGITS AFTER THE DECIMAL POINT

p^*	0.5		0.25		0.1	
Dist. Type	Even	Geo.	Even	Geo.	Even	Geo.
Ratio	2.88	1.88	3.00	1.94	3.00	1.94

baseline needs 1.88-3 times memory compared to the approach using a single VF.

E. Case Study A: 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 [13]. 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,



(a) absolute errors (b) relative errors

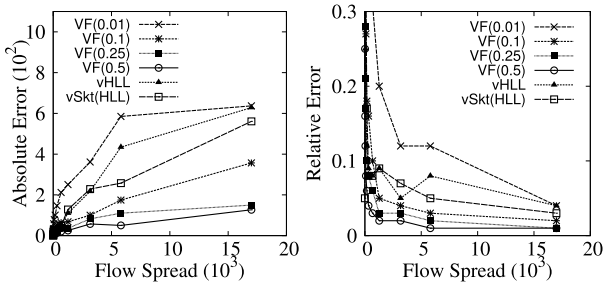
Fig. 8. Estimation accuracy of VF(p), vHLL, and vSkt(HLL) when all algorithms are allocated 2Mbits memory. The algorithm of VF(0.5) reduces average relative error by up to 80% and 78%, respectively, compared to vHLL and vSkt(HLL). When p is very small, the error of VF(p) becomes worse due to its increasing sampling error.

we only compare VF with the best sketches for flow spread measurement, i.e., vHLL [16] and vSkt(HLL) [17]. 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.

Each experiment in this case study (or the next one) is performed over all 10 traffic traces and we present the average results.

Estimation Accuracy: We first present the absolute error and relative error of all algorithms under 2Mbits memory, shown in Figures 8(a) and 8(b), respectively. The flows are placed in bins based on their actual spreads (which can be



(a) absolute errors (b) relative errors

Fig. 9. 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 2Mbits memory. The error of VF under 1 Mbits is very close to that under 2Mbits.

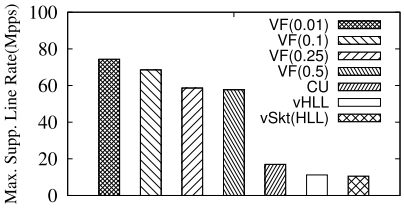


Fig. 10. Maximum supported line rate of VF(p) in comparison with CU, vHLL and vSkt(HLL). The maximum supported line rate of CU is 50% larger than vHLL and vSkt(HLL), and VF can improve the maximum supported line rate by over 2.29 times on the basis of CU.

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 be drawn from Figure 8(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 (1Mbits) and keeping the memory of vHLL and vSkt(HLL) as 2Mbits. The results in Figure 9 show that even if VF is allocated 1Mbits memory, its measurement accuracy is very similar to that under 2Mbits. 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 10. 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 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.

TABLE VIII
MEMORY REQUIREMENT (MBITS) OF VF AND VF+ UNDER DIFFERENT VALUES OF p

Alg. \ p	0.1	0.25	0.5
VF	0.48	1.22	1.80
VF+	0.31	0.78	1.14

F. Flow Spread Estimation for Dataset With Large Flows

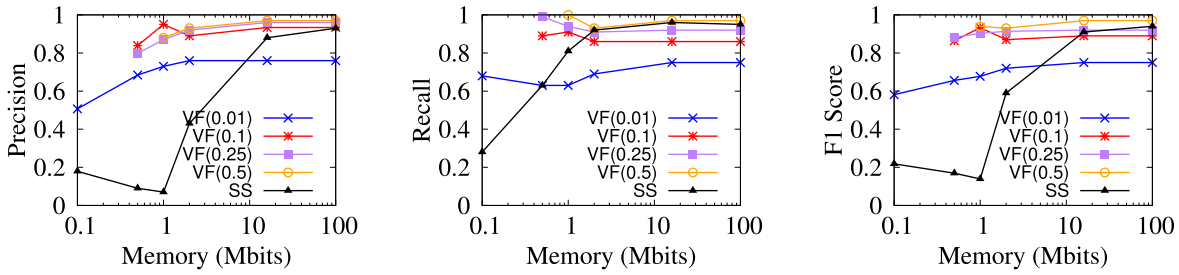
We simulate the scenario where there are several extreme large flows in the dataset by injecting multiple artificial large flows to the CAIDA dataset, which together consist of a new dataset, called CAIDA+. Two algorithms are compared. One is VF combined with CU for flow spread measurement method (see Section VI-A), called VF with confusion and the other is the optimized VF combined with CU to handle large flows (see Section VII), called VF+. The estimation results are plotted with scatter figure, where the x-axis is the actual spread and the y-axis is the estimate. Each dot in the figure represents a flow. We also plot the line $y = x$ in the figure. The closer to the line the dot is, the more accurate the estimate of the flow is. Both VF and VF+ employ a CU sketch with 1Mbits memory to deal with the sampled packets after the non-duplicate sampling. n_s is fixed as 7500 to accommodate the largest flow' spread.

We first give the memory consumption of VF and VF+ under different sampling probabilities $p = 0.5$, 0.25, 0.1. The results in Table VIII show that VF+ reduces 36% memory consumption over VF, regardless of the sampling probability, demonstrating that *the advantage of VF+ over VF is that its memory consumption is much smaller*. This advantage ensures that VF+ can support non-duplicated-sampling-enabled spread estimation with longer period compared to VF. We also give the results of maximum support n when memory allocation is 1Mbits. The results under different p in Table IX show that VF+ can extend the period of non-duplicate sampling by 35% compared to VF. This means that VF+ can remove duplicates by a longer period.

The estimation accuracy results are plotted in Figs. 12 and 13, respectively. Under the same sampling probability, visually, VF and VF produce similar spread estimates for flows. For statistically results, we divide the flows into different sets by their actual spreads. The spread ranges are $[1, 10)$, $[10, 10^2)$, $[10^2, 10^3)$, $[10^3, 10^4)$, $[10^4, \infty)$. Tbls. X, XI and XII give the average relative error for flows in different sets for VF and VF+ under the sampling probabilities of 0.5, 0.25, 0.1, respectively. Statistically, VF and VF+ have similar average absolute errors for different sets of flows, especially when the flows' spreads are within 10^4. This ensures that VF+ can maintain high accuracy as VF even with less memory.

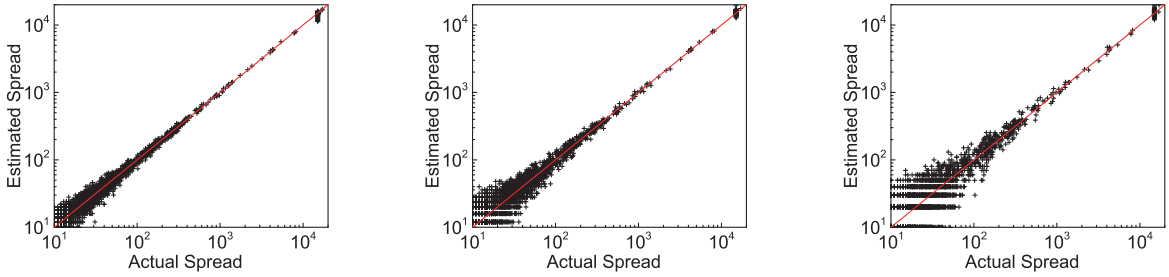
G. Case Study B: Super Spreader Detection

We investigate how VF+ performs in detection super spreaders in comparison with the state-of-the-art prior work, SpreadSketch(SS) [26]. As we have discussed in Section. VI-B, when detecting super spreaders, VF+ uses CMH to process sampled packets. Without loss of generality,



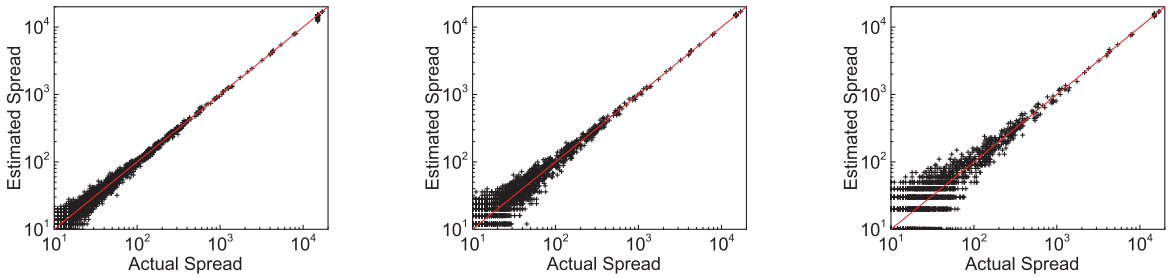
(a) Precision (b) Recall (c) F1-score

Fig. 11. Experimental results of precision, recall and f1-score of VF with different p in comparison with SS, under different memory allocations. VF and SS detect super spreaders accurately under large memory. Under tight memory, VF can maintain high detection accuracy, while SS cannot. The lines of VF under different p start from different memory in the figure as VF has minimum memory requirement that is related to p.



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

Fig. 12. Estimation accuracy of VF+ with different sampling probabilities under the CAIDA+ dataset.



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

Fig. 13. Estimation accuracy of VF with different sampling probabilities under the CAIDA+ dataset.

TABLE IX

MAXIMUM SUPPORTED n OF VF AND VF+ UNDER DIFFERENT VALUES OF p

Alg. \ p	0.1	0.25	0.5
VF	3.68	1.47	0.69
VF+	5.01	2.00	0.90

TABLE X

AVERAGE RELATIVE ERROR OF VF AND VF+ FOR FLOWS IN DIFFERENT SPREAD RANGES WITH $p = 0.5$

Alg. \ Spread	[1,10)	[10,10^2)	[10^2,10^3)	[10^3,10^4)	$\geq 10^4$
VF	3.08	0.18	0.06.	0.02	0.04
VF+	3.06	0.18	0.06	0.02	0.06

we also abbreviate it as VF+. The super spreader is defined as flows whose spreads exceed the threshold that the user chooses based on application need. In this experiment, we tune the threshold for each trace to keep the number of super spreaders in each trace as 100. The number of arrays for SS and CMH

TABLE XI

AVERAGE RELATIVE ERROR OF VF AND VF+ FOR FLOWS IN DIFFERENT SPREAD RANGES WITH $p = 0.25$

Alg. \ Spread	[1,10)	[10,10^2)	[10^2,10^3)	[10^3,10^4)	$\geq 10^4$
VF	3.01	0.31	0.10	0.03	0.01
VF+	3.06	0.31	0.10	0.03	0.06

TABLE XII

AVERAGE RELATIVE ERROR OF VF AND VF+ FOR FLOWS IN DIFFERENT SPREAD RANGES WITH $p = 0.1$

Alg. \ Spread	[1,10)	[10,10^2)	[10^2,10^3)	[10^3,10^4)	$\geq 10^4$
VF	2.90	0.54	0.17	0.05	0.02
VF+	2.88	0.53	0.18	0.06	0.09

are both set to 3 for fair comparison. We use the following five metrics.

- Precision: The ratio of true super spreaders detected over all true super spreaders existed.
- Recall: The ratio of true super spreaders detected over all super spreaders reported.
- F1-score: The harmonic average of precision and recall.

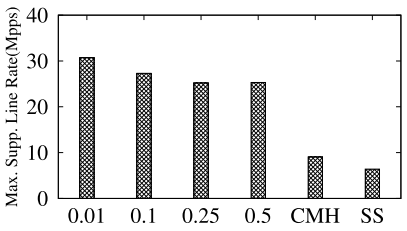


Fig. 14. Maximum supported line rate of VF+(p) (p shown in x-scale), in comparison with CMH and SS, for super spreader detection. CMH improves the maximum supported line rate by 50% compared to SS, and VF+ can improve the maximum line rate by over 178% compared to CMH.

- Relative error: It is defined in the previous subsection. We measure the average relative error of super spreaders reported by the algorithms.
- Maximum supported line rate. It has been defined before.

We conduct experiments under different memory allocation ranging from 0.1Mbits to 100Mbits. The experimental results are shown in Figure 11. Let memory be 16Mbits as an example, VF+(0.5) is the best algorithm, achieving near 1 of Precision, Recall and F1-score. SS can also maintain close-to-1 performance in terms of Precision, Recall and F1-score. Let's see Figure 11(a). SS's Precision is 0.43, which means near six out of ten super spreaders reported by SS are fake. In some scenarios, low Precision incurs a lot of additional false alarms and takes extra time from system admin to investigate.

Figure 14 compares the maximum supported line rate of each algorithm. The memory is 2Mbytes for each sketch. We stress that CMH is for heavy hitter identification while VF+(0.01), VF+(0.1), VF+(0.25), VF+(0.5) and SS are for super spreader detection. As we can see, CMH can improve the maximum supported line rate by 50% compared to SS. It shows the necessity of turning super spreader detection to heavy hitter identification. In addition, our VF+ can improve the maximum supported line rate by over 178% further compared to CMH.

IX. Conclusion and Future Work

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 sampling on multiple independent tasks with different sampling probabilities. We apply virtual filter on flow spread measurement and super spreader detection, and give a new design that can extend the measurement period when large flows. 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.

Our experimental studies have been software-based so far. Our future work will experiment with hardware implementation. VF only requires one hash operation and at most two memory accesses to process each data item (or packet), which makes it feasible for high-speed hardware implementation. Current PISA programmable switches use packet processing pipeline architecture where a function is split into several pipeline stages. While the number of stages, memory accesses and arithmetic operations is limited [41], VF can be implemented on a PISA programmable switch using only 2 pipeline stages. Step 1 and Step 2 can be combined into the first stage and Step 3 will be the second stage. The first stage has a hash operation, a modulus operation and two memory accesses (one read and one write). The hash operation and the memory accesses are well-supported by programming languages such as P4. In practice, if we set m' to power of 2, the modulus operation ($h(x) = H(x) \mod m'$) can be simplified as bit shifting. The comparison in the second stage is equivalent to $h(x) \times z < mm'p$, where the left side is an integer multiplication and the right side can be pre-computed. Therefore, VF will be suitable for implementation on a PISA programmable switch.

References

[1] C. Ma, H. Wang, O. O. Odegbile, and S. Chen, "Virtual filter for non-duplicate sampling," in *Proc. IEEE 29th Int. Conf. Netw. Protocols (ICNP)*, Nov. 2021, pp. 1–11.
[2] T. Benson, A. Anand, A. Akella, and M. Zhang, "MicroTE: Fine grained traffic engineering for data centers," in *Proc. 7th Conf. Emerg. Netw. Exp. Technol.*, 2011, pp. 1–12.
[3] 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 Trans. Netw.*, vol. 9, no. 3, pp. 265–279, Jun. 2001.
[4] Y. Xie, V. Sekar, D. A. Maltz, M. K. Reiter, and H. Zhang, "Worm origin identification using random moonwalks," in *Proc. IEEE Symp. Secur. Privacy*, May 2005, pp. 242–256.
[5] Cisco. *Cisco IOS NetFlow*. Accessed: 2022. [Online]. Available: http://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html
[6] Inmon Corporation. *sFlow Accuracy and Billing*. Accessed: 2022. [Online]. Available: https://inmon.com/technology/
[7] Z. Liu *et al.*, "Nitrosketch: Robust and general sketch-based monitoring in software switches," in *Proc. ACM Special Interest Group Data Commun.*, Aug. 2019, pp. 334–350.
[8] T. Yang *et al.*, "Elastic sketch: Adaptive and fast network-wide measurements," in *Proc. ACM SIGCOMM*, Aug. 2018, pp. 561–575.
[9] Q. Huang, P. P. C. Lee, and Y. Bao, "SketchLearn: Relieving user burdens in approximate measurement with automated statistical inference," in *Proc. SIGCOMM*, Aug. 2018, pp. 576–590.
[10] Q. Huang *et al.*, "SketchVisor: Robust network measurement for software packet processing," in *Proc. ACM SIGCOMM*, Aug. 2017, pp. 113–126.
[11] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman, "One sketch to rule them all: Rethinking network flow monitoring with UnivMon," in *Proc. ACM SIGCOMM Conf.*, Aug. 2016, pp. 101–114.
[12] G. Cormode and S. Muthukrishnan, "An improved data stream summary: The count-min sketch and its applications," *Proc. LATIN*, 2004, pp. 29–38.
[13] C. Estan and G. Varghese, "New directions in traffic measurement and accounting," in *Proc. ACM SIGCOMM*, Aug. 2002, pp. 323–336.
[14] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," in *Proc. Int. Colloq. Automata, Lang., Program. (ICALP)*, Jul. 2002, pp. 693–703.
[15] C. Ma, H. Wang, O. Odegbile, and S. Chen, "Noise measurement and removal for data streaming algorithms with network applications," in *Proc. IFIP Netw. Conf. (IFIP Networking)*, 2021, pp. 1–9.
[16] Q. Xiao, S. Chen, M. Chen, and Y. Ling, "Hyper-compact virtual estimators for big network data based on register sharing," in *Proc. ACM SIGMETRICS*, Jun. 2015, pp. 417–428.
[17] Y. Zhou, Y. Zhang, C. Ma, S. Chen, and O. O. Odegbile, "Generalized sketch families for network traffic measurement," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 3, no. 3, pp. 1–34, Dec. 2019.
[18] K.-Y. Whang, B. T. Vander-Zanden, and H. M. Taylor, "A linear-time probabilistic counting algorithm for database applications," *ACM Trans. Database Syst.*, vol. 15, no. 2, pp. 208–229, Jun. 1990.
[19] P. Flajolet and G. N. Martin, "Probabilistic counting algorithms for database applications," *J. Comput. Syst. Sci.*, vol. 31, pp. 182–209, Sep. 1985.
[20] P. Flajolet, E. Fusy, O. Gandouet, and F. Meunier, "HyperLogLog: The analysis of a near-optimal cardinality estimation algorithm," in *Proc. AOFA*, 2007, pp. 127–146.

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

[22] G. Cormode and S. Muthukrishnan, "Space efficient mining of multigraph streams," in *Proc. ACM PODS*, Jun. 2005, pp. 271–282.

[23] S. Ganguly, M. Garofalakis, R. Rastogi, and K. Sabnani, "Streaming algorithms for robust, real-time detection of DDoS attacks," in *Proc. 27th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, 2007, p. 4.

[24] W. Liu, W. Qu, J. Gong, and K. Li, "Detection of superpoints using a vector Bloom filter," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 3, pp. 514–527, Mar. 2016.

[25] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with opensketch," in *Proc. USENIX Symp. Netw. Syst. Design Implement.*, 2013, pp. 29–42.

[26] L. Tang, Q. Huang, and P. P. C. Lee, "SpreadSketch: Toward invertible and network-wide detection of superspreaders," in *Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM)*, Jul. 2020, pp. 1608–1617.

[27] S. K. Fayaz, Y. Tobioka, V. Sekar, and M. Bailey, "Bohatei: Flexible and elastic DDOS defense," in *Proc. 24th USENIX Secur. Symp. (USENIX Security)*, 2015, pp. 817–832.

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

[29] S. Singh, C. Estan, G. Varghese, and S. Savage, "Automated worm fingerprinting," in *Proc. OSDI*, vol. 4, p. 4, 2004.

[30] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, Jul. 1970.

[31] Y.-E. Sun, H. Huang, C. Ma, S. Chen, Y. Du, and Q. Xiao, "Online spread estimation with non-duplicate sampling," in *Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM)*, Jul. 2020, pp. 2440–2448.

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

[33] H. Huang *et al.*, "Spread estimation with non-duplicate sampling in high-speed networks," *IEEE/ACM Trans. Netw.*, vol. 29, no. 5, pp. 2073–2086, Oct. 2021.

[34] C. Estan, G. Varghese, and M. Fisk, "Bitmap algorithms for counting active flows on high speed links," in *Proc. ACM SIGCOMM Conf. Internet Meas.*, 2003, pp. 153–166.

[35] S. Venkataraman, D. Song, P. B. Gibbons, and A. Blum, "New streaming algorithms for fast detection of superspreaders," in *Proc. NDSS*, 2005, pp. 1–18.

[36] Y. Liu, W. Chen, and Y. Guan, "Identifying high-cardinality hosts from network-wide traffic measurements," *IEEE Trans. Depend. Sec. Comput.*, vol. 13, no. 5, pp. 547–558, Sep./Oct. 2016.

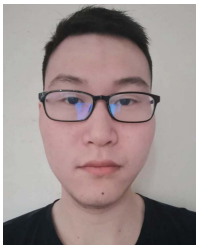
[37] Q. Xiao *et al.*, "Cardinality estimation for elephant flows: A compact solution based on virtual register sharing," *IEEE/ACM Trans. Netw.*, vol. 25, no. 6, pp. 3738–3752, Dec. 2017.

[38] S. Heule, M. Nunkesser, and A. Hall, "HyperLogLog in practice: Algorithmic engineering of a state of the art cardinality estimation algorithm," in *Proc. EDBT*, 2013, pp. 683–692.

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

[40] UCSD. (2015). *CAIDA UCSD Anonymized 2015 Internet Traces on Jan. 17*. [Online]. Available: http://www.caida.org/data/passive/passive_2015_dataset.xml

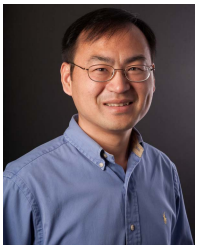
[41] R. Ben-Basat, X. Chen, G. Einziger, and O. Rottenstreich, "Efficient measurement on programmable switches using probabilistic recirculation," in *Proc. IEEE 26th Int. Conf. Netw. Protocols (ICNP)*, Sep. 2018, pp. 313–323.



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

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

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



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

Dimitrios Melissourgos received the B.E. degree from the Department of Computer Engineering and Informatics, University of Patras, and the master's degree in computer science from the Department of Computer and Information Science and Engineering, University of Florida, where he is currently pursuing the Ph.D. degree. His main research interests are networks traffic measurement and distributed machine learning privacy.