Distributed Non-Duplicate Sampling with Application on Network-wide Flow Cardinality Estimation

Aayush Karki[1] Zibo Liu[2] Shigang Chen[2] Haibo Wang[1]
[1]Department of Computer Science, University of Kentucky, USA
Email: {aka334, wanghaibo}@uky.edu
[2]Department of Computer & Information Science & Engineering, University of Florida, USA
Email: {ziboliu@ufl.edu, sgchen@cise.ufl.edu}

***Abstract*—Non-duplicate sampling (NDS) is a recently proposed technique that selects items from a data stream with probability p only on their first appearance, effectively handling duplicate items. While NDS has shown promise in the network application on flow cardinality measurement by outperforming sketch-based approaches, it faces limitations in distributed environments where traffic flows span multiple measurement points. To address this challenge, this paper proposes distributed non-duplicate sampling (DNDS), which extends NDS to multiple sampling points by ensuring each item is globally selected with probability p exactly once, regardless of where its appearances occur. We present the first comprehensive study of DNDS, focusing on its application on network-wide flow cardinality estimation. We develop an efficient implementation of DNDS with theoretical guarantees and derive optimal parameter settings. We propose two novel DNDS-based solutions for network-wide flow cardinality estimation. Through extensive experimental evaluation using real network traffic traces, we demonstrate the effectiveness of the DNDS implementation and demonstrate that our solutions achieve up to 10× improvement in estimation accuracy compared to state-of-the-art sketch-based methods.**

I. Introduction

Background: Consider an input data stream. *Traditional sampling* outputs a sub-stream by selecting each data item in the stream independently with a given probability p. Now suppose the stream is a multi-set, meaning that any item may appear multiple times. *Non-duplicate sampling* (NDS) selects each data item for its first appearance with probability p, while ignoring all subsequent appearances [1].

One application of NDS is flow cardinality estimation. Recent research advocates the implementation of network traffic measurement in real-time on the data plane [2], [3], [4], providing critical information that is essential to various network functions, from intrusion detection to performance optimization to service provision. One important measurement is on the number of *distinct* packets in each flow, called *flow cardinality*, where two packets are distinct (or identical) if their attributes of interest are different (or same). For example, consider per-destination flows where all packets to the same destination address form a flow, with the source address of each packet being the attribute. In this case, a flow's cardinality is the number of distinct sources that send packets to the flow's destination. Monitoring this information for all destinations in a network helps identify potential DDoS attacks [5]. For another example, consider per-source flows where all packets from the same source address form a flow, with the destination address of each packet being the attribute. A flow's cardinality is the number of distinct destinations that the flow's source send packets to. Monitoring this information for all sources in a network helps identify potential scanners [5].

Flow cardinality measurement requires us to remember all attributes that have been seen in each flow so that duplicates can be removed, which however requires excessive memory for implementation on the data plane. Most solutions to this problem use sketches, which are compact, probabilistic data structures that provide estimates instead of precise counts. The sketches for estimating the cardinality of a single flow include bitmap [6], FM [7], multi-resolution bitmap (MRB) [8], KMV [9], LogLog [10], HyperLogLog (HLL) [11], [12], and self-morphing bitmap [13], each requiring hundreds or more bits in order to handle a large cardinality range. For millions of flows, it is not practical to assign each flow a separate sketch on the data plane. Hence, the multi-flow solutions must share sketches among the flows, which however degrades the accuracy of cardinality estimation [14], [5], [15], [16], [17], [18], [19], [3], [20], [21], [2].

This is where NDS comes in [1], [22], [23], [24], [25], [26]: Each packet is modeled as a data item, consisting of a flow label and an attribute; for per-destination flows, the flow label is destination address and the attribute is the source address. Because NDS removes all duplicates, it makes the problem of counting distinct packets much easier. It also performs sampling, allowing tradeoff between counting accuracy and processing overhead. The NDS solutions for cardinality estimation have demonstrated superior performance compared to their sketch-based counterparts [22], [23], [24].

New Problem: This paper studies a new problem of *distributed non-duplicate sampling* (DNDS), with an application on *network-wide cardinality estimation*. Consider multiple data streams, each at a separate sampling point. The same item may appear multiple times and in different streams. The problem is to perform distributed and cooperative sampling at those points such that each data item is *globally* selected with probability p for one appearance while all other appearances are ignored, regardless of their locations. Exactly one appearance of each item globally will have a probability p of being selected, and all other appearances have zero chance.

DNDS matches well to network-wide cardinality estimation, where each flow may be distributed at multiple measurement points across a network. Consider a network with multiple gateways for robust Internet access. Suppose a traffic measurement module is deployed at each gateway to monitor per-destination flow cardinality in the inbound traffic for potential DDoS attacks or monitor per-source flow cardinality in the outbound traffic for potential scanners in its network. Performing NDS at each gateway independently cannot remove all duplicates because duplicate packets (from the same source to the same destination) could pass different gateways and therefore they may be selected for more than once. Solving this problem will require DNDS, which promises to remove duplicates globally.

Contribution: This paper is the first to study DNDS. To demonstrate its practicality, we will use network-wide flow cardinality as the context to describe our solution. The contributions are threefold:

First, we define a new type of sampling called distributed non-duplicate sampling (DNDS), with application on network-wide flow cardinality estimation. We develop an efficient algorithm to implement DNDS, which requires only one hash per packet. We present theoretical analysis to prove its correctness and derive optimal parameters.

Second, we propose DNDS-based solutions for network-wide flow cardinality estimation under two different models. The first model is an on-chip-off-chip model, where DNDS is deployed in on-chip memory to process packets in real time. The sampled packets, with a significantly lower rate than the original packet stream, are processed in off-chip memory for better estimation accuracy. The second model is an on-chip model, where both DNDS and post-DNDS processing are deployed in on-chip memory for real-time estimation in the data plane.

Third, we implement the new sampling algorithm and the two solutions for network-wide flow cardinality estimation. Trace-driven experimental results demonstrate that the new sampling algorithm is highly efficient and that the DNDS-based solutions reduce estimation error by up to 10× compared to the state-of-the-art sketch-based solutions for cardinality estimation.

II. Network-wide Flow Cardinality Estimation

We introduce the problem of network-wide flow cardinality estimation, which is used as the application context for distributed non-duplicate sampling in the next section.

A. System Model

We consider a network with u measurement points, denoted as $\mathcal{P}_0, \mathcal{P}1, \ldots, \mathcal{P}_{u-1}$, which may be gateways, routers or switches, each implementing a traffic measurement module. Each measurement point $\mathcal{P}_i$, $0 \leq i < u$, receives a packet stream S_i. Let the *grand packet stream* be $G = \bigcup_{i=0}^{u-1} S_i$.

Tasks	Flow label	Attribute
Super spreaders	src IP	dstIP
(stealthy) DDoS attacks	dstIP	srcIP
Port scan	srcIP-dstIP	dstPort
Heavy hitters	srcIP-dstIP	timestamp
SYN-flood	srcIP-srcPort-dstIP-dstPort	timestamp
Stealthy scanner	srcIP	dstIP-dstPort

TABLE I: Examples of flow cardinality definitions under various network tasks.

Consider the following example, where $u = 3$ and there are four packets, x_0, x_1, x_2 and x_3, whose routing paths are $\mathcal{P}_0 \rightarrow \mathcal{P}_1$, $\mathcal{P}_1 \rightarrow \mathcal{P}_2$, $\mathcal{P}_0 \rightarrow \mathcal{P}_2$, and $\mathcal{P}_1$, respectively. The packet streams at the measurement points are $S_0 = \{x_0, x_2\}$, $S_1 = \{x_0, x_1, x_3\}$, and $S_2 = \{x_1, x_2\}$. The grand stream is $G = \{x_0, x_0, x_1, x_1, x_2, x_2, x_3\}$.

B. Flow Model

Each packet x is modeled as a pair $x = \langle f, e \rangle$, where f is a flow label and e is an attribute. We define a flow f as the set of packets that share the same flow label f. The label may be the five-element tuple for TCP. It may also be 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. The attribute e is typically chosen as a value or a combination of values from the packet headers or payload.

C. Network-wide Flow Cardinality Estimation

Two packets are distinct if they belong to different flows or their attributes are different; otherwise, they are identical. For a packet stream, a flow's cardinality is defined as the number of *distinct* packets in the flow. For a grand packet stream, a flow's network-wide cardinality is its number of distinct packets across all measurement points. The problem of network-wide flow cardinality estimation is to provide an estimate for the network-wide cardinality of any given flow.

Continue the example in Section II-A. Assume that all the packets belong to flow f. Let $x_0 = \langle f, e_0 \rangle$, $x_1 = \langle f, e_0 \rangle$, $x_2 = \langle f, e_1 \rangle$, and $x_3 = \langle f, e_2 \rangle$. Packets x_0 and x_1 are identical. The grand packet stream G is $\{\langle f, e_0 \rangle, \langle f, e_0 \rangle, \langle f, e_0 \rangle, \langle f, e_0 \rangle, \langle f, e_1 \rangle, \langle f, e_1 \rangle, \langle f, e_2 \rangle\}$. The network-wide cardinality of f is 3 because there are three distinct attributes e_0, e_1, e_2 in the flow. In contrast, the packet streams at the measurement points, $\mathcal{P}_0$, $\mathcal{P}_1$ and $\mathcal{P}_2$, are $S_0 = \{\langle f, e_0 \rangle, \langle f, e_0 \rangle\}$, $S_1 = \{\langle f, e_0 \rangle, \langle f, e_0 \rangle, \langle f, e_2 \rangle\}$, and $S_2 = \{\langle f, e_0 \rangle, \langle f, e_1 \rangle\}$, respectively. The cardinalities of flow f at $\mathcal{P}_0$, $\mathcal{P}_1$, and $\mathcal{P}_2$ are 1, 2, and 2, respectively. In general, the measurement points cannot individually provide accurate network-wide flow cardinality estimation. They need to work together, distributedly.

Some different flow types and the network tasks that their cardinalities may support can be found in Table I.

III. Distributed Non-duplicate Sampling

A. Challenges for Network-wide Cardinality Estimation

To estimate the network-wide flow cardinality for any flow, it is necessary to eliminate duplicate packets within the flow. There are two types of duplicates:

- *Intra-point duplicates*. These are packets with the same flow label and the same attribute that may appear multiple times at the same measurement point. These packets should be counted only once in the flow cardinality measurement. In the example of Section II-C, $\mathcal{P}_0$ has an intra-point duplicate packet $\langle f_0, e_0 \rangle$.
- *Inter-point duplicates*. These occur when different measurement points observe packets with the same flow label and the same attribute, resulting in duplicates across measurement points. To accurately measure the network-wide flow cardinality, we need to keep only one of these packets and disregard the others. In the example of Section II-C, $\langle f_0, e_0 \rangle$ is an inter-point duplicate that appears at three points $\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2$.

While the traditional sampling methods fail to address both intra-point and inter-point duplicates, recent non-duplicate sampling methods [1], [22], [23], [24], [25], [26] successfully eliminate intra-point duplicates. NDS achieves this by sampling a packet with a fixed probability only on its first appearance and ignoring all subsequent occurrences, thereby producing duplicate-free samples that accurately represent the distribution of distinct packets in the original stream. However, NDS falls short for inter-point duplicates because it is designed for a single, localized stream, whereas network-wide measurement involves geographically distributed packet streams across multiple measurement points.

B. Distributed Non-duplicate Sampling

To remove the inter-point duplicates, we propose *distributed non-duplicate sampling* (DNDS) as follows:

Definition 1: Given a sampling probability p and a grand packet stream G consisting of multiple geographically distributed packet streams, for any packet x in G, DNDS should sample it with probability p for one of its appearances and block all other appearances.

Note that DNDS does not necessarily sample the packet with probability p at its first appearance. For the purpose of counting cardinality, any appearance is equally fine, as long as only one appearance is sampled with probability p.

We will use our DNDS algorithm as a packet pre-processing step for network-wide flow cardinality estimation. DNDS pre-processes the incoming packet stream at line rate, producing a sub-stream of sampled packets at each measurement point. This sub-stream is then sent to the traffic measurement module, where flow cardinality measurement is performed. The details of the post-sampling measurement module will be discussed in Section V.

C. Performance Metrics

Any algorithm that implements DNDS will need a data structure to record the packets that have been seen. Such a data structure will have a limited recording capacity, i.e., the expected number of distinct packets it can record, which is determined by the amount of allocated memory. We define the *sampling period* as the expected number n of distinct packets that the DNDS algorithm can process before its data structure becomes saturated and can no longer ensure DNDS. After this period, we must start a new period and initialize the data structure. Therefore, DNDS is achieved for the grand packet stream within each period.

x	Packet
$x_1, ..., x_{10}$	Packet stream in the example
p	Sampling probability
$\mathcal{P}_i$	ith measurement point
B_i	Bitmap at $\mathcal{P}_i$
u	Number of measurement points
G	Grand packet stream
n	Number of distinct packets in G
S_i	Local packet stream passing through $\mathcal{P}_i$
$H_i(\cdot)$	Uniform hash functions
$h(\cdot)$	Uniform hash function
$H'_i(\cdot)$	Uniform hash functions
B_i^j	jth segment in B_i
l^j	Number of bits in B_i^j
q_j	Number of bis of $B_i^0, ..., B_i^j$
$I_{h(d)}$	The index of segment that bit $B_i[h(d)]$ locates at
z_j	Number of zero bits in segment B_j^j

TABLE II: Notations.

Besides correctness, the performance of a DNDS algorithm is evaluated by three metrics: (1) Given a sampling period of n distinct packets, it should use as little on-chip memory as possible; (2) given a memory allocation, it should extend its sampling period as long as possible; (3) its per-packet processing overhead should be as small as possible in order to support highest possible line rate.

IV. DNDS Algorithm Design

We begin with a naive solution and then present our design. Some important notations are given in Table II.

A. A Naive Solution

The idea is to ensure that each packet is always processed at the same measurement point. Specifically, each packet, regardless of where and when it appears, is hashed and then redirected to a specific measurement point. This approach effectively divides the grand packet stream into multiple mutually exclusive packet streams, each assigned to one measurement point. As a result, there is no inter-point duplicate, allowing us to apply the existing NDS solution at each point.

However, the downside of this solution is that $\frac{u-1}{u}$ packets are expected to be redirected from one point to another, leading to significant bandwidth consumption.

B. Our Solution

We aim to optimize the naive solution by reducing the excessive packet redirection. Recognizing the need for communication among measurement points to distribute the information that a packet has arrived—thereby blocking future duplicates of that packet at other points—our goal is to minimize such communications. Our idea is to introduce a local filtering process to remove the intra-point duplicates. Most packets will not pass through this local filtering, and only those packets likely to be making their first global appearance will pass through and be further sent to the measurement point to which the packet is hashed for inter-point duplicate removal. We will conduct experiments to verify that such overhead is very small in Table V and Section VI-B; only a tiny fraction of packets need to be redirected, and only their flow labels and attributes are sent to other measurement points, while the original packets will proceed as usual.

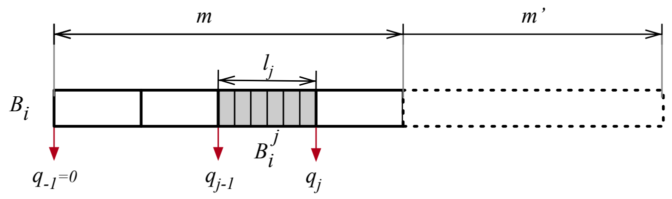


Fig. 1: The data structure B_i of DNDS at any measurement point $\mathcal{P}_i$. B_i is an array of m' bits, where only the first m bits are real. The real part can also be split to u segments B_i^0, B_i^1,... B_i^{u-1}.

Data structure and algorithm. As shown in Figure 1, each measurement point is equipped with a bitmap B_i of m' bits, with only the first m bits being real. We call $B_i[0]\dots B_i[m-1]$ the real part of the filter and $B_i[m]\dots B_i[m'-1]$ the virtual part of the filter. The m-bit real part is divided into u segments, $B_i^0, B_i^1, B_i^2, \dots, B_i^{u-1}$, each corresponding to a measurement point. Let the number of bits in B_i^j be l^j and the cumulative number of bits of the first j segments be q_j, i.e.,

$$q_j = \sum\nolimits_{k=0}^{j} l^k. \tag{1}$$

Let q_{-1} be 0. The l^j bits in segment B_i^j, i.e., $B_i^j[0], B_i^j[1], \dots, B_i^j[l^j - 1]$, correspond to $B_i[q_{j-1}], B_i[q_{j-1}+1], \dots, B_i[q_j - 1]$ in B_i.

Each time a packet x is received at measurement point $\mathcal{P}_i$, we compute the hash $h(x) = H(x) \mod m'$, where $H(x)$ is a hash function with a range sufficiently larger than m'. We then perform the following four steps:

Step 1: Sampling with probability $\frac{m}{m'}$. If $h(x) \geq m$, it falls in the virtual part of the filter, and we ignore the packet, which does not cause any memory overhead or further processing overhead since recording does not happen for this packet. If $h(x) < m$, it falls in the real part of the filter, and we proceed to the next step.

Step 2: Filtering local duplicates. If $B_i[h(x)]$ is set to one, we do nothing further, and the packet is blocked. Otherwise, we set $B_i[h(x)] = 1$, find the segment that $B_i[h(x)]$ belongs to, denoted as $B_i^{I_{h(x)}}$, where $I_{h(x)}$ satisfies

$$q_{I_{h(x)}-1} \leq h(x) < q_{I_{h(x)}}. \tag{2}$$

We then send the packet x to measurement point $\mathcal{P}_{I_{h(x)}}$, and proceed to the next step. This packet redirection overhead should be minimized. Essentially, only the flow label and attribute of x are sent to point $\mathcal{P}_{I_{h(x)}}$ instead of the raw packet. Moreover, we will conduct experiments to verify that only a very small portion of packets will be redirected in Table V and Section VI-B.

Step 3: Filtering global duplicates. At measurement point $\mathcal{P}_{I_{h(x)}}$, if $B_{I_{h(x)}}[h(x)]$ is one, we do nothing further, and the packet is blocked. This occurs either because the packet x is not the first appearance in G or due to other packets setting the bit through hash collisions. Since we cannot differentiate between these scenarios, we safely discard the packet. If $B_{I_{h(x)}}[h(x)]$ is zero, we assert that packet x must be the first appearance in G and proceed to the next step.

Step 4: Compensating sampling with increasing probability to achieve a total sampling probability p. Let $z_{I_{h(x)}}$ be the number of zeros in segment $B_{I_{h(x)}}$. At this stage, knowing x must be the first appearance, we can calculate the probability that x is mapped to a zero bit in segment $B_{I_{h(x)}}$ as $\frac{m}{m'} \cdot \frac{l^{I_{h(x)}}}{m} \cdot \frac{z_{I_{h(x)}}}{l^{I_{h(x)}}} = \frac{z_{I_{h(x)}}}{m'}$. This probability decreases as $z_{I_{h(x)}}$ decreases over time. To maintain a fixed total sampling probability of p, we need a *compensating sampling* with an increasing probability of $\frac{m' p_{I_{h(x)}}}{z_{I_{h(x)}}}$, where $p_k = \frac{l^k}{m} p$ represents the target probability in our design that any packet at its first appearance gets sampled by exactly $\mathcal{P}_k$, $0 \leq k < u$. Specifically, we generate a random number $r \in [0, 1)$ and output x if $r < \frac{m' p_{I_{h(x)}}}{z_{I_{h(x)}}}$. Afterward, we set $B_{I_{h(d)}}[h(x)] = 1$ to block subsequent appearances of the packet. Moreover, our algorithm terminates when $\frac{m' p_k}{z_k} > 1$ occurs at any point $\mathcal{P}_k$.

The formal description is provided in Algorithm 1.

Example. Consider a network with three measurement points $\mathcal{P}_0$, $\mathcal{P}_1$, $\mathcal{P}_2$, and a grand packet stream containing 10 distinct packets, $x_1, x_2, \dots, x_{10}$, ordered by arrival time as shown in Fig. 2.

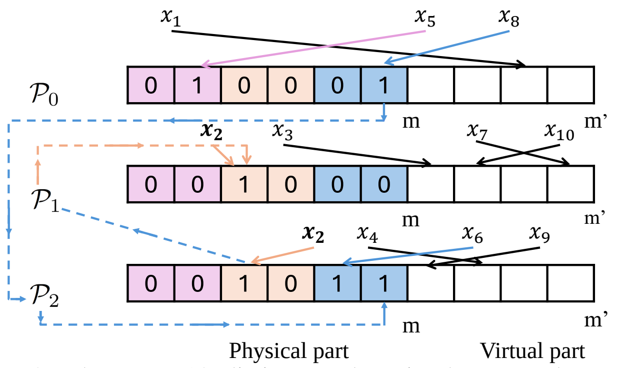


Fig. 2: There are 10 distinct packets in the example grand packet stream. Among these packets, x_2, x_5, x_6, and x_8 are hashed to the real part of the bitmap. The other packets are hashed to the virtual part and are thus blocked. The first appearance of x_2, x_5 and x_8 are sampled. The second appearance of x_2 is blocked by Step 3 because the bit it hashes to has been set to one by the first appearance of x_2. Packet x_6 passes through Steps 1-3 but is not sampled in Step 4.

- x_1: Hashed to the virtual part at $\mathcal{P}_0$ and ignored in Step 1.

Algorithm 1 Distributed non-duplicate sampling with probability p on any switch $\mathcal{P}_i$

```
1: Input: sampling probability $p_i$, number of total distinct packets tend to process in a period: $n$
2: Action: perform distributed non-duplicate sampling
3: // setting $m, m'$ according to Theorem 2
4: if $p < \frac{1}{e}$ then
5:     $m = npe, m' = n$
6: else
7:     $m = -\frac{n}{\ln p}, m' = -\frac{n}{\ln p}$
8: end if
9: // data structure initialization
10: for Each measurement point $\mathcal{P}_i$ do
11:     create a bitmap $B_i$ of $m$ bits, set $z_i = l^i$
12: end for
13: //packet recording at any measurement point $\mathcal{P}_j$
14: for each packet $x$ arriving at $\mathcal{P}_j$ do
15:     $h(x) = H(x) \mod m'$
16:     if $h(x) < m$ then
17:         if $B[h(x)] = 0$ then
18:             $B[h(x)] = 1$
19:             $target\_switch\_id = I_{h(x)}$
20:             if $target\_switch\_id \neq j$ then
21:                 forward $x$ to $target\_switch\_id$
22:             else
23:                 $z_j = z_j - 1$
24:                 generate a random number $r \in [0, 1)$
25:                 if $r < \frac{m'l^j p}{mz_j}$ then
26:                     $x$ is sampled
27:                 end if
28:                 if $\frac{m'l^j p}{mz_j} > 1$ then
29:                     terminate sampling
30:                 end if
31:             end if
32:         end if
33:     end if
34: end for
```

- x_2 **(first arrival)**: Hashed to the real part at $\mathcal{P}_1$. The bit is zero (which will be set to one to block future duplicates), indicating its first appearance. With $I_{h(x_2)} = 1$, no packet redirection is needed. After passing through Step 4, it gets sampled by DNDS.
- x_3: Hashed to the virtual part at $\mathcal{P}_1$ and is ignored.
- x_2 **(second arrival)**: Hashed to the real part at $\mathcal{P}_2$ and its hashed bit is zero (which will be set to one to block future duplicates), indicating the first appearance of this packet at $\mathcal{P}_2$. With $I_{h(x_2)} = 1$, the packet is forwarded to $\mathcal{P}_1$ and further ignored by $\mathcal{P}_1$ as the packet already appeared.
- x_4: Hashed to the virtual part at $\mathcal{P}_2$ and is ignored.
- x_5: Hashed to the real part at $\mathcal{P}_0$ and its hashed bit is zero. So it passes through Steps 1 and 2. With $I_{h(x_5)} = 0$. It automatically passes through Step 3. After passing through Step 4, it gets sampled by DNDS.
- x_6: Hashed to the real part at $\mathcal{P}_2$ and its hashed bit is zero. So it passes through Steps 1 and 2. With $I_{h(x_6)} = 2$. It automatically passes through Step 3. However, it does not pass through Step 4, and thus is not sampled by DNDS.
- x_7: Hashed to the virtual part at $\mathcal{P}_1$ and is ignored.
- x_8: Hashed to the virtual part at $\mathcal{P}_0$, and its hashed bit is zero. So it passes through Steps 1 and 2. With $I_{h(x_8)} = 2$. x_8 is redirected to $\mathcal{P}_2$, where the $h(x_8)$th bits is zero. Thus, the packet passes through Step 3. It automatically passes through Step 3. After passing through Step 4, it gets sampled by DNDS.
- x_9, x_{10}: Hashed to the virtual part and are thus ignored.

C. Analysis

Correctness proof. For correctness, any packet will pass the filter with probability p at its first appearance and will be blocked for subsequent appearances, which is proven in the following Theorem 1.

Theorem 1: Algorithm 1 will sample each distinct packet once with probability p, regardless how many times it appears and where it appears.

Proof: For any packet x that appears at arbitrary measurement point $\mathcal{P}_i$, the probability for it to pass through Step 1 (which means x is hashed to the real part of B_i) is $\frac{m}{m'}$. We combine the process in Steps 2 and 3 together. Packet x passes through Steps 2 and 3 only when its hashed bit in switch $\mathcal{P}_i$, i.e., $B_i[h(x)]$ and its hash bit switch $\mathcal{P}_j$, i.e., $B_j[h(x)]$ are both zero, assuming that $I_{h(x)} = j$ for ease of representation. Note that if $B_j[h(x)] = 0$, $B_i[h(x)] = 0$ must holds as otherwise the packet that sets $B_i[h(x)] = 1$ must be redirected to switch $\mathcal{P}_j$ and set $B_j[h(x)] = 0$. Therefore, we only need to care about the $B_j[h(x)]$ when calculating the probability of passing through Steps 2 and 3, which is $\frac{z_j}{l^j} \cdot \frac{l^j}{m} = \frac{z_j}{m}$. The sampling probability in Step 4 is $\frac{p_j m'}{z_j}$. Therefore, the total sampling probability of passing through Steps 1-4 under the condition of $I_{h(x)} = j$ is $\frac{m}{m'} \cdot \frac{z_j}{m} \cdot \frac{p_j m'}{z_j} = p_j$. Considering that the packet can be hashed to any segment, i.e., $I_{h(x)} = 0, 1, ..., u-1$, The total sampling probability is $\sum_{j=0}^{u-1} = p_j$.

For the first appearance of any packet that is redirected to $\mathcal{P}_{I_{h(x)}}$, it will be sampled with probability of p. For the second appearance of that packet that is redirected to $\mathcal{P}_{I_{h(x)}}$, its hash bit $B_{I_{h(d)}}[I_{h(x)}] = 1$ must hold as at least this bit must have been set by its first appearance. Therefore, duplicates are removed, all those appearances will not pass the filter. ∎

Optimal parameter setting. The following theorem guarantees the optimal parameter setting under given p and n.

Theorem 2: Let n be the expected number of distinct packets to be processed in the grand packet stream G. The optimal parameter setting of Algorithm 1 is

$$m' = \begin{cases} n, & p < \frac{1}{e} \\ -\frac{n}{\ln p}, & \frac{1}{e} \leq p < 1 \end{cases} \quad m = \begin{cases} npe, & p < \frac{1}{e} \\ -\frac{n}{\ln p}, & \frac{1}{e} \leq p < 1 \end{cases} \tag{3}$$

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 packets, the expected number of packets recorded in the real part of all B_i is $n\frac{m}{m'}$. The algorithm terminates when $\frac{m' p_k}{z_k} = 1$ at any point $\mathcal{P}_k$, which is equivalent to $\frac{z_k}{m} = \frac{pm' l^k}{m^2}$. Since each packet will be randomly hashed to any bit in any segments. According to [27], the expected number of packets recorded in the real part of all B_i is $-m \ln \frac{\sum_{k=0}^{m-1} z_k}{m}$, which is $-m \ln \frac{pm'}{m}$, combining with $\frac{z_k}{m} = \frac{pm' l^k}{m^2}$, 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 packets n and the number of bits m are usually very large and n/m is a constant by (3). According to Alg. 1, a sampling period ends when $z = m'p$. At that time, the expected number of packets recorded in the bitmap should not be less than $n\frac{m}{m'}$. Therefore, we have $n\frac{m}{m'} \leq -m \ln \frac{m'p}{m} \Rightarrow \ln \frac{m'p}{m} \leq -\frac{n}{m'} \Rightarrow m \geq m'pe^{\frac{n}{m'}}$. The minimum value of m is achieved when $m = m'pe^{\frac{n}{m'}}$. Taking the first-order derivative on the right side, we have $\frac{\mathrm{d}m}{\mathrm{d}m'} = \frac{\mathrm{d}m'pe^{\frac{n}{m'}}}{\mathrm{d}m'} = pe^{\frac{n}{m'}} - \frac{np}{m'}e^{\frac{n}{m'}} = e^{\frac{n}{m'}}p(1 - \frac{n}{m'})$

Setting $\frac{\mathrm{d}m}{\mathrm{d}m'} = 0$, we have $m' = n$. Besides, when $m' < n$, $\frac{\mathrm{d}m}{\mathrm{d}m'} < 0$; when $m' > n$, $\frac{\mathrm{d}m}{\mathrm{d}m'} > 0$. Therefore, the minimum value of m, is npe which achieves when $m' = n$. However, since $m \leq m'$, this parameter setting is valid only when $p \leq \frac{1}{e}$. For $p > \frac{1}{e}$, we always have $m' > n$ from (3) and $\frac{\mathrm{d}m}{\mathrm{d}m'} > 0$, which means the optimal setting is $m' = m$. Under this condition, we have $m' = m = m'pe^{\frac{n}{m'}} \Rightarrow 1 = pe^{\frac{n}{m'}} \Rightarrow m' = -n/\ln p$ In this case, we have $m' = -\frac{n}{\ln p}$. ■

Performance analysis. The following theorem proves the maximum support grand packet stream size under given memory allocation and p.

Theorem 3: Given a distributed non-duplicate sampling probability p and a memory allocation of m bits at each measurement point, the expected number of distinct packets in the grand packet stream that can be recorded before starting the next sampling period is

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

The proof is trivial and thus omitted. It can be derived easily from (3).

V. Application: Network-wide Flow Cardinality Measurement with DNDS

A. On-Chip-Off-Chip Model: DNDS + Hash Table

We propose an on-chip-off-chip model for network-wide flow cardinality measurement, where the DNDS solution is implemented in on-chip memory such as SRAM, and the hash table is implemented in off-chip memory to store the counts of samples in each flow. The advantage of this model is that it utilizes the high speed of on-chip memory to match the line rate of packet streams and the large capacity of off-chip memory. This model can be especially applied for offline data analytics.

After DNDS processing, duplicates in the packet streams are removed, and a subset of distinct packets is sampled. The sampled packets will be output at a significantly reduced rate. For example, the CAIDA 1-minute traffic trace contains 20 million packets but only 430 thousand distinct source-destination IP pairs, resulting in a duplicate ratio of 46.5.

Furthermore, the on-chip-off-chip model ensures that communication between the on-chip SRAM and off-chip hash table is one-way: only packets sampled by the on-chip DNDS are forwarded to the off-chip hash table, with no reverse communication.

To perform network-wide flow cardinality measurement, we deploy a hash table HT_i in off-chip memory at each measurement point $\mathcal{P}_i$. For any packet $\langle f, e \rangle$, if it is sampled by DNDS with probability p, we perform insertion with the following two cases.

- If HT_i does not contain f in the key set, we insert entry $\langle f, \frac{1}{p} \rangle$ to HT_i
- Otherwise, we increment the value of f by $\frac{1}{p}$, that is $HT_i.put(f, HT_i.getValue(f) + \frac{1}{p})$.

For query on flow f's cardinality, we sum up the value of f in all hash table HT_0, HT_1, ..., HT_{m-1}. That is, we return

$$\hat{n}_f = \sum_{i=0}^{u-1} HT_i.getOrDefault(f, 0) \tag{5}$$

B. Entirely On-Chip Model: DNDS + CU Sketch

The entirely on-chip model utilizes another property of DNDS: The sampled packets after DNDS contains no duplicates and each distinct packet has the same probability p of being sampled. Therefore, we can count the number of packets in the samples for each flow, called *flow size*. The network-wide flow cardinality can be derived by dividing the flow size by p.

We can adopt the compact data structures, called sketches to measure flow sizes in the samples. The well-known sketches are CM [28], CU [29], and CS [30], with CU being recognized as the most accurate. Therefore, it is adopted in this paper. We first review the CU sketch:

CU sketch. The data structure is a two-dimension array of C of counters, with k rows and w columns. The jth counter in the ith row is denoted as $C_i[j]$, with $0 \leq i < k$, $0 \leq j < w$. Each counter is initialized to zero. There are two operations.

•*Recording*: We record a packet $\langle f, e \rangle$ to k counters $C_0[H'_0(f)]$, $C_1[H'_1(f)]$,..., $C_{k-1}[H'_{k-1}(f)]$, where $H'_i(\cdot) \in [0, w-1]$, with $0 \leq i < k$ are k independent uniform hash functions. Let the minimum value among the k hash counters as $v_{\min}$, i.e.,

$$v_{\min} = \min_{0 \leq i < k} C_i[H'_i(f)]. \tag{6}$$

For each counter $C_i[H'_i(f)]$ with $0 \leq i < k$, CU increases its value to $v_{\max} + \frac{1}{p}$ if its value is smaller than $v_{\max} + \frac{1}{p}$, i.e., $C_i[H'_i(f)] = \max\{C_i[H'_i(f)], v_{\max} + \frac{1}{p}\}$.

Actually, in implementation, if $v_{\max} + \frac{1}{p}$ is not integer, we will increase $C_i[H'_i(f)]$ to $\lceil v_{\max} + \frac{1}{p} \rceil$ with probability of $\frac{v_{\max} + \frac{1}{p} - C_i[H'_i(f)]}{\lceil v_{\max} + \frac{1}{p} \rceil - C_i[H'_i(f)]}$.

• *Look-up*: Return the minimum value as the estimate of flow f, i.e., $\min_{0 \leq i < k} C_i[H'_i(f)]$.

To perform network-wide flow cardinality measurement, we deploy a CU sketch C^j in on-chip memory at each measurement point $\mathcal{P}_j$ For any packet $\langle f, e\rangle$, if it is sampled by DNDS with probability p, we record it to C^j based on the recording operation of CU.

For query on flow f's cardinality, we sum up the estimate of f from all measurement points by performing look-ups in all CU sketches C^0, C^1, ..., C^{u-1}. That is, we return

$$\hat{n}_f = \sum\nolimits_{0\leq j<u} \min_{0\leq i<k} C_i^j[H_i'(f)]. \tag{7}$$

The entirely on-chip model enables real-time queries of network-wide flow cardinality [19]. For example, when applications need to identify flows with rapidly increasing cardinalities to detect potential DDoS attacks in real-time, DNDS + CU can perform the look-up operation in (7). Since this operation is already executed during packet recording (6), DNDS + CU incurs minimal additional overhead when responding to online queries.

VI. Experimental Evaluation

A. Experimental Setting

We have implemented the proposed algorithm for DNDS. We have also implemented (1) two DNDS-based solutions for network-wide flow cardinality estimation, DNDS + Hash Table and DNDS + CU, and (2) the state-of-the-art prior work that performs network-wide flow cardinality estimation, vSkt(HLL) [5], rSkt2(HLL) [31], AROMA [21]. The experiments are performed on a computer with 13th Generation Intel Core i7-13700 (30MB Cache, 16 Core (8+8), 2.1GHz to 5.2GHz (65W)) and 16 GB memory.

The data traces used in our evaluation are real Internet traffic traces downloaded from CAIDA [32]. We use 10 traces, each containing around 20M packets. Each experiment is performed over 10 traces and we present the average results. Flow label is defined as destination address and the attribute is the source address, both carried in each packet's header, which has the application of DDoS detection. Flow cardinality is the number of distinct sources that communicate with a destination. Packets will be distinct if they possess different flow labels or attributes. Each trace contains around 430k distinct packets, i.e., n ≈430k. The CAIDA data set is used as the grand packet stream. For each packet we will randomly assign it to one or multiple measurement points.

The parameters settings of the proposed algorithm, 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, while m' is size of the whole bitmap, including the virtual part. Therefore, we use m to denote the memory allocation of the proposed DNDS algorithm. The number of measurement points are set as $u = 20$. We follow the parameter settings of vSkt(HLL), rSkt2(HLL), AROMA, in the original papers. Specifically, The HLL register is 5bits and each flow is mapped to 128 registers for vHLL and vSkt(HLL). Each entry in AROMA consists of a 32-bit key field and a 32-bit value field. Each measurement point will be deployed with an instance of the measurement algorithm with the same parameter setting.

p	0.5	0.25	0.1	0.01
Throughput (Mpps)	70.2	64.5	62.4	59.8

TABLE III: Maximum supported line rate of the proposed algorithm under different sampling probabilities p.

m(Mbits)	0.1		1		10	
	$\hat{n}$	n	$\hat{n}$	n	$\hat{n}$	n
0.5	0.07	0.07	0.69	0.69	6.92	6.93
0.25	0.15	0.15	1.46	1.47	14.64	14.71
0.10	0.36	0.37	3.62	3.67	36.23	36.79
0.01	3.62	3.67	36.23	36.78	362.34	367.87

TABLE IV: Maximum number of distinct packets that can be supported by the proposed algorithm $\hat{n}$ ($\times 10^6$) vs. theoretical maximum supported n by Eq. (4) under different sampling probabilities p, and memory allocations m.

B. Sampling Performance of DNDS

This paper is the first to define and implement DNDS. We evaluate the sampling performance of the proposed algorithm in the following four dimensions.

Actual sampling rate: We evaluate the actual sampling rate of ten random chosen subsets, each containing around 50k distinct packets, under given sampling probability p= 0.01, 0.1, 0.25 and 0.5, respectively. The results in Figure 3 show that the actual sampling rate is close to p, especially when p is large.

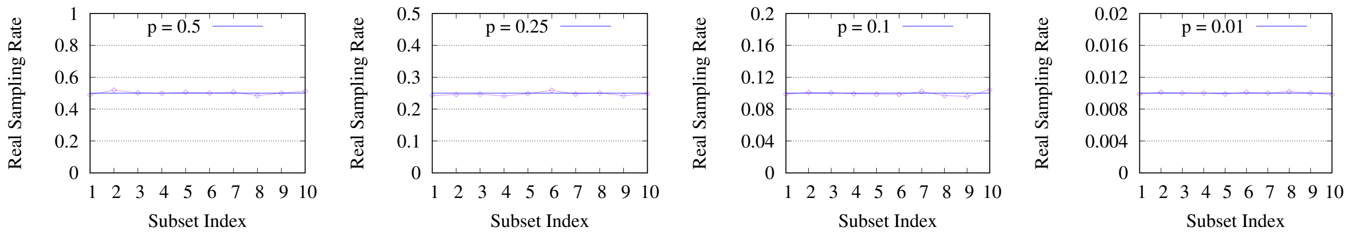
Maximum supported line rate: We measure the maximum line rate the DNDS solution can catch up when processing packet streams. The unit is million packets per second, abbreviated as Mpps. Mpps is changed to Gbps if multiplying the average packet size (kbits) in the trace — if the average packet size is 1kbits, 1Mpps and 1Gbps represent the same maximum supported line rate. The results in Table III shows that the proposed algorithm can support processing at least 59.8 million packets per second.

Maximum supported $\hat{n}$: It is defined as the maximum number of distinct packets the proposed algorithm can support under given sampling probability p and memory allocation m. Large maximum n means longer sampling period. We can also calculate the theoretical n by Eq. (4). The results are shown in Table IV, where we can see the $\hat{n}$ and n are very similar.

Ratio of redirected packets: It is defined as the ratio of the number of redirected packets from one measurement point to another in line 21 of Algorithm 1 over the total number of packets in the packet stream. The results are shown in Table V, where we observe only small portion of packets whose flow labels and attributes will be redirected. The ratios are 2.5%, 1.5%, $< 1\%$ and <0.1% under the probabilities of 0.5, 0.25, 0.1, and 0.01, respectively.

C. Network-wide Flow Cardinality Estimation using DNDS

In this section, we evaluate the performance of the proposed network-wide flow cardinality estimation algorithm using two approaches in Section V: DNDS + Hash Table and DNDS + CU under different memory configurations and sampling



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

Fig. 3: Actual sampling rate w.r.t. subset index, under different given sampling probability p. The difference between actual sampling rate and p is within $0.02p$, when $p \geq 0.1$, and within $0.05p$ when $p = 0.01$.

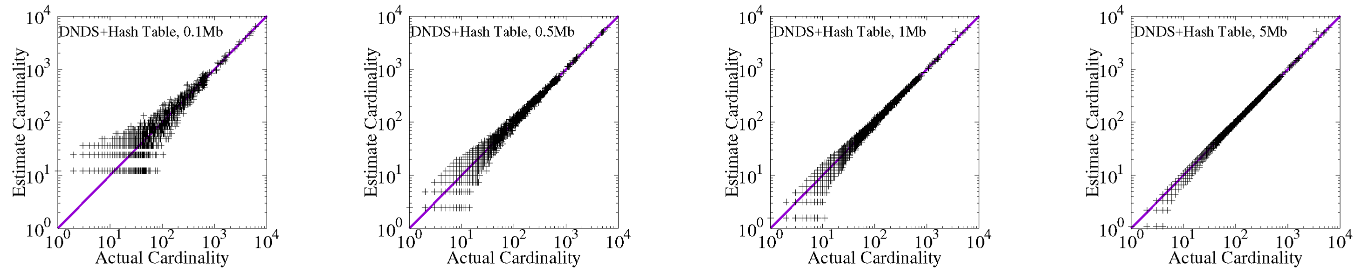


Fig. 4: Scatter plots: Estimated cardinality vs. actual cardinality of DNDS + Hash Table for network-wide flow cardinality estimation under the memory allocations of 0.1Mb, 0.5Mb, 1Mb, and 5Mb, respectively.

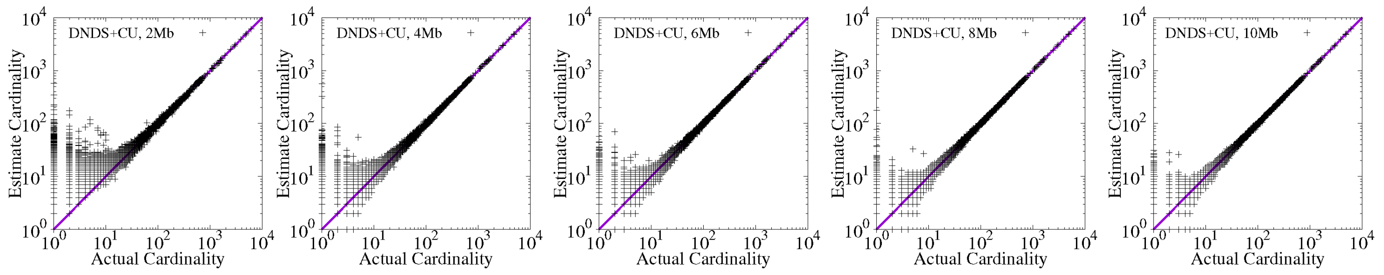


Fig. 5: Scatter plots: Estimated cardinality vs. actual cardinality of DNDS + CU for network-wide flow cardinality estimation, under the memory allocations of 2MB, 4Mb, 6Mb, 8Mb, and 10Mb, respectively, respectively.

p	0.5	0.25	0.1	0.01
Ratio	0.02451	0.01517	0.00607	0.00072

TABLE V: Ratio of redirected packets under different sampling probabilities.

m(Mbits)	p	(0, 10]	$(10, 10^2]$	$(10^2, 10^3]$	$(10^3, 10^4]$	$(10^4, 10^5]$
0.1	0.0825	2.3	12.3	42.7	125.0	605.3
0.5	0.4100	1.3	4.5	14.4	38.4	161.0
1.0	0.6403	0.8	2.8	9.3	27.4	55.2
5.0	0.9147	0.2	1.2	3.9	10.9	34.6

TABLE VI: Absolute error of network-wide flow cardinality estimation with DNDS + Hash Table, under different memory allocations.

m(Mbits)	p	(0, 10]	$(10, 10^2]$	$(10^2, 10^3]$	$(10^3, 10^4]$	$(10^4, 10^5]$
2	0.6897	10.3	5.0	9.8	30.5	72.7
4	0.8305	4.0	2.4	7.2	18.5	25.7
6	0.8835	2.2	1.9	6.2	13.9	41.3
8	0.9113	1.4	1.6	5.3	11.0	17.7
10	0.9284	1.0	1.4	5.0	9.7	6.3

TABLE VII: Absolute error of network-wide flow cardinality estimation with DNDS+CU, under different memory allocations.

probabilities. We use the absolute error as the accuracy metric, defined as $\sum |s_f - \hat{s}_f|/N$, where $\hat{s}_f$ and s_f are the estimated and actual cardinality of flow f, respectively, and N is the number of flows in the flow set.

In addition to the average error as the accuracy metrics, we employ scatter plots to comprehensively present first-hand estimation results. In scatter plots, each dot represents a flow with the x-axis being the real value and the y-axis being the estimate. The flows are placed in bins based on their actual cardinalities (which can be found directly from the traffic traces). The cardinality bins are (0,10], $(10^1, 10^2]$, $(10^2, 10^3]$, $(10^3, 10^4]$, and $(10^4, 10^5]$. We average the absolute/relative error of flows in each bin.

DNDS + Hash Table: The memory allocated for this method varied from 100 kbits to 5 Mbits. The sampling probabilities were calculated from Theorem 2 based on the memory allocation and the total number of distinct packet n

in the data set. The results, as shown in Table VI, indicate that increasing the memory allocation reduces both the absolute error across all bins. For instance, when the memory is increased from 100 kbits to 5 Mbits, the absolute error in the largest bin range (10^4, 10^5] decreases from 605.3 to 34.6. This demonstrates the effectiveness of higher memory allocations in improving the accuracy of flow cardinality estimation. We also plot the scatter plot under these memory allocations, shown in Figure 4, where we can visually find all dots concentrates to the line $y = x$, especially under 1Mb and 5Mb memory allocations where precise estimation is achieve for every flow.

DNDS + CU: Both DNDS and CU are implemented in on-chip memory, therefore both account for the total memory allocation. Define the *partition ratio* as the ratio of memory for the DNDS over the total memory. The larger the partition ratio is, the more memory allocated to DNDS and the less to CU. After that, the respective sampling probability can be derived from the memory allocated to DNDS and n. We vary the total memory allocations from 2 Mbits to 10 Mbits, a narrower range compared to the memory allocation to DNDS + Hash Table as a workable CU sketch needs at least million bits. We want to stress that we have conducted exhaustive preliminary experiments to find a good setting of the partition ratio. Given memory from 1 Mbits to 10 Mbits, we test the accuracy performance of flows in different bins, under different partition ratios from 0.1 to 0.9 with a step length of 0.1. The results are very detailed and are omitted due to space limit. But the conclusion is that the partition ratio is recommended to be 0.6 for memory sizes in [1Mbits, 10Mbits]. Moreover, the accuracy performance is not sensitive to the partition ratio. A slight difference from 0.6 can also achieve good performance. In the rest of this section, we will set the partition ratio as 0.6.

The absolute error results for DNDS + CU are presented in Table VII. Similar to the hash table-based method, increasing the total memory allocation reduces both the absolute and relative errors across all bin ranges. For example, with a memory allocation of 10 Mbits, the absolute error in the largest bin range (10^4, 10^5] is 6.33, such small values that precise estimation can be achieved. We also plot the scatter plot under these memory allocations, shown in Figure 5. we can visually find all dots concentrates to the line $y = x$, especially under 8Mb and 10Mb memory allocations where almost all small flows' estimates deviates within 50.

D. Accuracy Comparison with Sketch-based Solutions

Our solutions, DNDS + Hash Table and DNDS + CU are the first sampling-based solutions for network-wide flow cardinality measurement. Existing solutions are sketch-based where each measurement point will be deployed a sketch in the on-chip memory and will be aggregated for network-wide flow cardinality queries offline. We compare our solutions with the state-of-the-art sketches that can support network-wide measurement, i.e., vSkt(HLL) [5], rSkt2(HLL) [31], and AROMA [21]. We inherit the accuracy metrics of absolute error and scatter plots in the previous Section VI-C.

Scatter plot presentation under 1Mb memory (Figure 6). In each figure, we also plot line $y = x$. From the scatter plots, we find DNDS + Hash Table is visually the most accurate sketch. Its dots concentrate to the line $y = x$ most than other solutions. DNDS + CU outperforms all sketches in terms of the accuracy of large flows whose cardinalities are above 10^2. Its dots for small flows are visually slightly more concentrated than vSkt(HLL). The dots of AROMA, vSkt(HLL), and rSkt2(HLL) are more scattered, especially for those numerous dots for small flows. AROMA only tracks partial flows as it is designed for measuring large flows. In the plot, there are fewer dots than other sketches' plots. In addition, even if small or medium flows get sampled by AROMA, their estimation error is large, around 10^2 in Figure 6.

Scatter plot presentation under 5Mb memory (Figure 7). When memory increases, we can visually find the dots are more concentrated to the line $y = x$ for each solution. But we emphasize that DNDS + Hash Table dots are still more concentrated than others: almost all dots are in the line $y = x$. We can also find that DNDS + CU's dots are visually more concentrated than rSkt2(HLL) and vSkt(HLL), especially for medium and large flows. AROMA still cannot measure all small/medium flows.

Average absolute error under 1Mb memory (Table VIII). We give the statistical results of average absolute error for flows whose real cardinalities are in the same range. The ranges are $(0, 10^1]$, $(10^1, 10^2]$, $(10^2, 10^3]$... For Table VIII, DNDS + Hash Table is the only one that can achieve the average absolute error $< 10^1$ for flows in the ranges of $(0, 10^1]$, $(10^1, 10^2]$ and $(10^2, 10^3]$. Other solutions' error is larger than 10^1. For medium and large flows, DNDS + Hash Table and DNDS + CU are the best two solutions, and are the only two solutions can achieve the average absolute error < 100 for flows whose cardinalities are above 100.

Average absolute error under 5Mb memory (Table IX). DNDS + Hash Table and DNDS + CU maintain their best accuracy over other solutions. Specifically, DNDS + Hash Table reduces the average absolute error by 96.8%, 97.3%, and 95.8%, , compared to AROMA, rSkt2(HLL), and vSkt(HLL), for flows in the range of $(0, 10^1]$, and by 91.6%, 97.0%, and 94.6% compared to AROMA, rSkt2(HLL), and vSkt(HLL), for flows in the range of $(10^4, 10^5]$, respectively. DNDS + CU reduces the average absolute error by 53.4%, 61.5%, and 38.6%, , compared to AROMA, rSkt2(HLL), and vSkt(HLL), for flows in the range of $(0, 10^1]$, and by 88.8%, 95.0%, and 84.7% compared to AROMA, rSkt2(HLL), and vSkt(HLL), for flows in the range of $(10^4, 10^5]$, respectively.

VII. Related Works

Traditional sampling and non-duplicate sampling: A simple sampling technique is systemic sampling, where a packet is deterministically selected based on arrival time or its position in the data stream. Examples of applications/protocols that implement systemic sampling are sFlow [33], NetFlow [34] and Juniper per-packet sampling. Another sampling technique

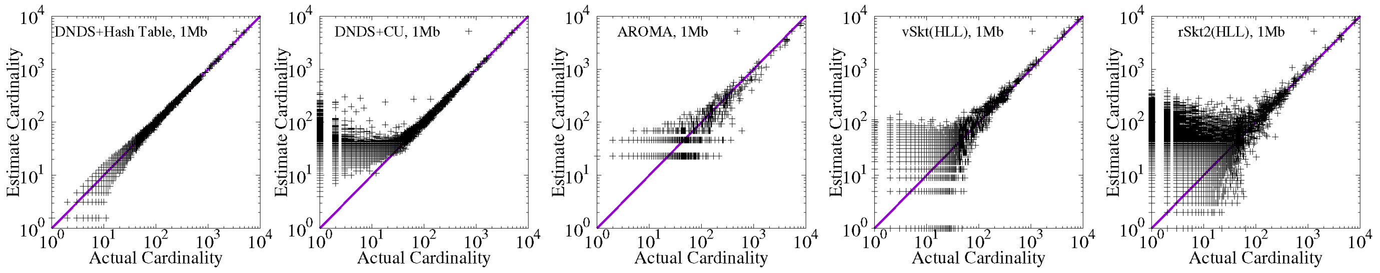


Fig. 6: Scatter plots: Estimated cardinality vs. actual cardinality of DNDS + Hash Table, DNDS + CU, AROMA, vSkt(HLL), rSkt2(HLL) for network-wide flow cardinality estimation under the memory allocations of 1Mb.

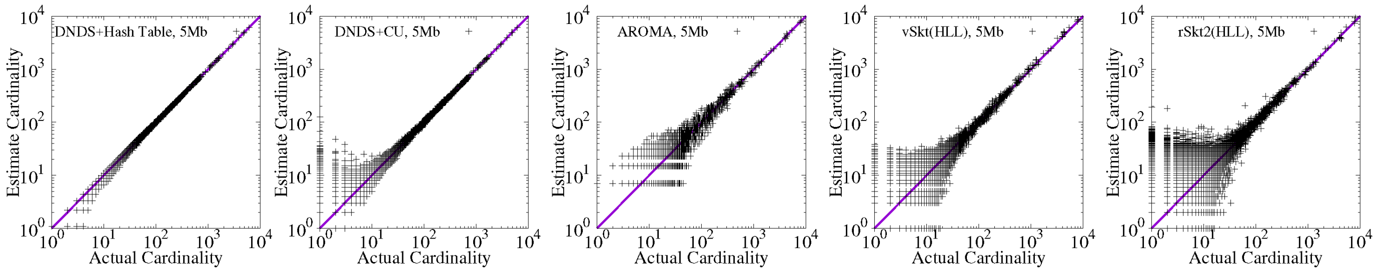


Fig. 7: Scatter plots: Estimated cardinality vs. actual cardinality of DNDS + Hash Table, DNDS + CU, AROMA, vSkt(HLL), rSkt2(HLL) for network-wide flow cardinality estimation under the memory allocations of 5Mb.

Cardinality	Flows	DNDS+HT	DNDS+CU	AROMA	rSkt2(HLL)	vSkt(HLL)
(0, 10]	109558	0.8	24.8	31.1	22.3	12.7
(10, 10^2]	2848	2.8	14.9	25.9	23.1	14.1
(10^2,10^3]	281	9.3	13.9	64.2	33.7	35.5
(10^3,10^4]	23	27.4	31.0	325.1	278.9	375.7
(10^4,10^5]	2	55.2	99.0	879.5	1945	644

TABLE VIII: Average absolute error of flows in different ranges under the CAIDA dataset. Memory for each solution is 1Mb. This table presents the statistical results in Figure 6.

Cardinality	Flows	DNDS+HT	DNDS+CU	AROMA	rSkt2(HLL)	vSkt(HLL)
(0, 10]	109558	0.2	2.9	6.2	7.5	4.7
(10, 10^2]	2848	1.2	2.1	9.6	8.1	5.7
(10^2,10^3]	281	3.9	7.1	32.9	25.2	20.4
(10^3,10^4]	23	10.8	24.2	95.8	245.1	256.1
(10^4,10^5]	2	34.6	31.7	407.4	1133.8	640.7

TABLE IX: Average absolute error of flows in different ranges under the CAIDA dataset. Memory for each solution is 5Mb. This table presents the statistical results in Figure 7.

is a random sampling where a number of packets are randomly selected from total packets. Non-duplicate sampling can support flow cardinality measurement. Bloom filter [35] can remove duplicates but it cannot support the same sampling probability. The first work that supports the same sampling probability for each distinct packet is two-phase protocol for non-duplicate sampling [1] proposed by Sun et al.. Some subsequent work [23], [24], [25], [26] further optimize the performance of non-duplicate sampling or the flow cardinality estimation. But they are designed for single measurement point and cannot deal with network-wide flow cardinality estimation due to duplicates among measurement points.

Sketch-based flow cardinality measurement: To measure the flow cardinality for each flow in the packet stream, CSE [36] and vHLL [37] reduce memory consumption through space sharing. bSketch [5] and vSketch [5] propose a family of sketches using plug-ins like bitmaps [27], FM sketches [7] and HLL sketches [11], [12]. Among all the solutions, vSketch using HLL sketches, denoted as vSkt(HLL) is the most accurate one. rSkt2 is a generalized sketch framework using plug-ins such as bitmaps [27], FM sketches [7] and HLL sketches [11], [12]. The most accurate one is rSkt2(HLL). vSkt(HLL) and rSkt2(HLL) are designed originally for single measurement point but can also support network-wide flow cardinality estimation. AROMA [21] is specially designed for network-wide flow cardinality estimation. SpreadSketch [38] is designed to only estimate the network-wide cardinality of large flows; most of the small flows are ignored.

VIII. Conclusion

This paper proposes a new sampling called distributed non-duplicate sampling (DNDS), which can sample each distinct packet at its first appearance with probability p and block its subsequent appearances at any measurement points. We propose an algorithm to implement DNDS and theoretically analyze its performance and optimally set the parameters. We then propose two DNDS-based solutions to network-wide flow cardinality estimation, which outperforms existing sketch-based solutions significantly in terms of accuracy.

Acknowledgment

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

References

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

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

[3] H. Wang, C. Ma, O. O. Odegbile, S. Chen, and J.-K. Peir, "Randomized error removal for online spread estimation in high-speed networks," *IEEE/ACM Transactions on Networking*, 2022.

[4] Q. Xiao, Y. Cai, Y. Cao, and S. Chen, "Accurate and O (1)-Time Query of Per-Flow Cardinality in High-Speed Networks," *IEEE/ACM Transactions on Networking*, 2023.

[5] 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.

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

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

[8] C. Estan, G. Varghese, and M. Fisk, "Bitmap Algorithms for Counting Active Flows on High-Speed Links," *IEEE/ACM Transactions on Networking*, vol. 14, no. 5, pp. 925–937, 2006.

[9] K. Beyer, P. Haas, B. Reinwald, Y. Sismanis, and R. Gemulla, "On Synopses for Distinct-value Estimation under Multiset Operations," in *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, 2007, pp. 199–210.

[10] M. Durand and P. Flajolet, "Loglog counting of large cardinalities," *European Symposia on Algorithms*, pp. 605–617, 2003.

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

[12] S. Heule, M. Nunkesser, and A. Hall, "HyperLogLog in Practice: Algorithmic Engineering of a State of the Art Cardinality Estimation Algorithm," in *Proceedings of the 16th International Conference on Extending Database Technology*. ACM, 2013, pp. 683–692.

[13] H. Wang, C. Ma, S. Chen, and Y. Wang, "Online cardinality estimation by self-morphing bitmaps," in *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 2022, pp. 1–13.

[14] Y. Li, R. Miao, C. Kim, and M. Yu, "FlowRadar: A Better NetFlow for Data Centers," *in Proc. of USENIX NSDI*, 2016.

[15] Q. Xiao, S. Chen, M. Chen, and Y. Ling, "Hyper-compact virtual estimators for big network data based on register sharing," in *Proceedings of the 2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, 2015, pp. 417–428.

[16] M. Yoon, T. Li, S. Chen, and J.-K. Peir, "Fit a spread estimator in small memory," in *IEEE INFOCOM 2009*. IEEE, 2009, pp. 504–512.

[17] ——, "Fit a compact spread estimator in small high-speed memory," *IEEE/ACM Transactions on Networking*, vol. 19, no. 5, pp. 1253–1264, 2010.

[18] Q. Xiao, S. Chen, Y. Zhou, M. Chen, J. Luo, T. Li, and Y. Ling, "Cardinality estimation for elephant flows: A compact solution based on virtual register sharing," *IEEE/ACM Transactions on Networking*, vol. 25, no. 6, pp. 3738–3752, 2017.

[19] H. Wang, C. Ma, O. O. Odegbile, S. Chen, and J.-K. Peir, "Randomized error removal for online spread estimation in data streaming," *Proceedings of the VLDB Endowment*, vol. 14, no. 6, 2021.

[20] R. Ben-Basat, G. Einziger, S. L. Feibish, J. Moraney, B. Tayh, and D. Raz, "Routing-oblivious network-wide measurements," *IEEE/ACM Transactions on Networking*, vol. 29, no. 6, pp. 2386–2398, 2021.

[21] R. B. Basat, X. Chen, G. Einziger, S. L. Feibish, D. Raz, and M. Yu, "Routing oblivious measurement analytics," in *2020 IFIP Networking Conference (Networking)*. IEEE, 2020, pp. 449–457.

[22] H. Huang, Y.-E. Sun, C. Ma, S. Chen, Y. Du, H. Wang, and Q. Xiao, "Spread estimation with non-duplicate sampling in high-speed networks," *IEEE/ACM Transactions on Networking*, vol. 29, no. 5, pp. 2073–2086, 2021.

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

[24] C. Ma, H. Wang, O. O. Odegbile, S. Chen, and D. Melissourgos, "Virtual filter for non-duplicate sampling with network applications," *IEEE/ACM Transactions on Networking*, vol. 30, no. 6, pp. 2818–2833, 2022.

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

[26] Y. Du, H. Huang, Y.-E. Sun, S. Chen, G. Gao, X. Wang, and S. Xu, "Short-term memory sampling for spread measurement in high-speed networks," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 2022, pp. 470–479.

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

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

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

[30] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," in *International Colloquium on Automata, Languages, and Programming*. Springer, 2002, pp. 693–703.

[31] H. Wang, C. Ma, O. O. Odegbile, S. Chen, and J.-K. Peir, "Randomized Error Removal for Online Spread Estimation in Data Streaming," *Proceedings of the VLDB Endowment*, vol. 14, no. 6, pp. 1040–1052, 2021.

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

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

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

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

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

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

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