DUNE: Improving Accuracy for Sketch-INT Network Measurement Systems

Zhongxiang Wei, Ye Tian*, Wei Chen, Liyuan Gu, and Xinming Zhang
Anhui Key Lab on HPC, School of Computer Science and Technology, University of Science and Technology of China
Email: {wz199758, szcw33, guliyuan1}@mail.ustc.edu.cn, {yetian, xinming}@ustc.edu.cn

IEEE INFOCOM 2023 - IEEE Conference on Computer Communications | 979-8-3503-3414-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/INFOCOM53939.2023.10229098

***Abstract*—In-band Network Telemetry (INT) and sketching algorithms are two promising directions for measuring network traffic in real time. To combine sketch with INT and preserve their advantages, a representative approach is to use INT to send a switch sketch in small pieces (called *sketchlets*) to end-host for reconstructing an identical sketch. However, in this paper, we show that when naively selecting buckets to add to sketchlets, the end-host reconstructed sketch is inaccurate. To overcome this problem, we present *DUNE*, an innovative sketch-INT network measurement system. DUNE incorporates two key innovations: First, we design a novel *scatter sketchlet* that is more efficient in transferring measurement data by allowing a switch to select individual buckets to add to sketchlets; Second, we propose lightweight data structures for tracing "freshness" of the sketch buckets, and present algorithms for smartly selecting buckets that contain valuable measurement data to send to end-host. We theoretically prove the effectiveness of our proposed methods, and implement a prototype on commodity programmable switch. Results from extensive experiments driven by real-world traffic suggest that DUNE can substantially improve the measurement accuracy at a trivial cost.**

***Index Terms*—Network measurement, sketch, In-band Network Telemetry (INT), programmable switch**

I. Introduction

Faults and errors are inevitable in today's production networks, and how to monitor network health in real time is a critical problem. With the advances of software-defined networking (SDN) and data plane programmability, a number of measurement-based solutions have been proposed for troubleshooting networks in recent years. Among them, one promising direction is In-band Network Telemetry (INT) [1]–[7]. In INT, a switch piggybacks network states in packet header, and sends them to end-host for analysis. The benefit of INT is its speed, as network states can be collected at line rate. However, by carrying measurement data, INT consumes extra bandwidth, thus considerably impacts an INT flow's goodput and completion time [4]–[7].

Another promising direction is sketching algorithms [8]–[16]. In a sketch-based network measurement system, a probabilistic data structure, namely *sketch*, is maintained by a switch for aggregating per-flow statistics. Sketch is generally flexible as it allows a tradeoff between accuracy and memory usage. However, to send sketches to analyzers, the system either demands a dedicated out-of-band channel [8], or has to compress the sketches (and consequently sacrifices accuracy) to adapt to the available shared bandwidth [11].

To combine sketch with INT and to preserve their advantages, there are two representative approaches. One approach, with SketchINT [17] as an example, is to "construct sketch at end-host". In such an approach, a network switch piggybacks measurement data with an INT flow, and when receiving INT packets, the end-host aggregates the per-packet information into a group of sketches. The advantage of this approach is that end-host can maintain novel and complex sketches that are difficult to be implemented on switch hardwares. However, it does not avoid the large network overhead of INT, as per-packet data is directly transferred without being aggregated within the network.

The other approach, with LightGuardian [18] as an example, is to "reconstruct sketch at end-host". In this approach, a sketch is maintained by a programmable switch for tracing per-flow traffic statistics. In addition, the switch splits the sketch structure into many small pieces, called *sketchlets*, and an INT flow embeds the sketchlets in its packet headers to transfer to end-host, where the sketchlets are assembled for reconstructing a sketch that has an identical structure as the one on the switch.

The "reconstructing sketch at end-host" approach effectively reduces the INT bandwidth usage, as a sketchlet contains flow-level statistics rather than per-packet information. However, as we will see in this paper, the end-host reconstructed sketch is not as accurate as the switch sketch, for the reason that the two sketches can not be timely synchronized; and when data in the reconstructed sketch is invalid or stale, inaccuracy arises.

In this paper, we present *DUNE*, a sketch-INT network measurement system. DUNE employs the "reconstructing sketch at end-host" approach and improves the measurement accuracy by making two key innovations: First, we propose a novel sketchlet named *scatter sketchlet*, which is more efficient in transferring measurement data to end-host; Second, we develop lightweight data structures and algorithms for selecting sketch buckets that contain "fresh" measurement data to add to sketchlets. We carefully design the data structures and algorithms under the constraints of the RMT (Reconfigurable Match-Action Tables) programmable switch architecture [19], [20], and implement a prototype with a commodity Tofino-based switch. More specifically, we make the following contributions in this paper:

- We present a novel sketchlet design named *scatter sketch-*

*Corresponding author.

This work was supported in part by the National Natural Science Foundation of China (Nos. 61672486 and 62072425), in part by the SPR program of the Chinese Academy of Sciences (No. XDC02070300), and in part by the Fundamental Research Funds for the Central Universities (No. KY0110000053).

let, which allows a switch to select individual sketch buckets to add to sketchlets. We prove in theory that the scatter sketchlet is more efficient in transferring measurement data to end-host than the existing approach.
- We design lightweight data structures for tracing "freshness" of measurement data in sketch buckets, and present algorithms for selecting buckets that contain valuable measurement data to send to end-host. We theoretically prove that our method achieves the desired property in selecting sketch buckets at frequencies proportional to the buckets' update frequencies, therefore better synchronizes the end-host reconstructed sketch with the switch sketch.
- We realize our proposed data structures and algorithms on a commodity P4-programmable Tofino switch under the device's stringent register access constraints.
- We carry out extensive experiments with DUNE, and find that the system substantially improves measurement accuracy, at a trivial cost on switch's forwarding performance.

The remainder part of this paper is organized as follows. We discuss the related works in Sec. II. Sec. III explains our motivation and gives an overview of the system. Sec. IV presents the design and analysis of the scatter sketchlet. We propose the sketch bucket selection algorithms in Sec. V and describe the prototype implementation in Sec. VI. Sec. VII presents the experiment results and we conclude in Sec. VIII.

II. Related Work

With the advances of software-defined networking (SDN) and programmable data plane in recent years, In-band Network Telemetry (INT), which piggybacks measurement data in packet header, becomes a promising direction for measuring large-scale networks. Over the OpenFlow data plane, PathDump [3] traces per-packet trajectory and provides a set of APIs for network debugging. Jeyakumar et al. [1] propose to allow programmable switches to execute "tiny packet programs" (TPPs) embedded in packets to collect per-packet network states. Kim et al. [2] demonstrate that INT can be realized on P4-programmable data plane. A major concern of INT is that by carrying measurement data in packet header, INT consumes extra bandwidth, and its overhead increases with network size. To reduce the INT overhead, Sheng et al. [4] present DeltaINT, which reduces the INT overhead by selectively carrying network states only when their values change significantly. Tang et al. [21] propose Sel-INT for maximizing the information gain of INT and minimizing the bandwidth overheads at same time. Basat et al. [6] present PINT, which applies various probabilistic techniques to encode measurement data on multiple packets, so as to reduce the per-packet INT overhead. Chen et al. [7] propose to optimally schedule multiple PINT tasks within a fixed INT overhead.

Sketching algorithm is another promising direction. In a sketch-based measurement system, usually a probabilistic data structure, namely sketch, is maintained by a switch for aggregating per-packet information. Representative sketches include bitmap [22], hashing table [23], count-min [24], Bloom filter [25], and their variants. Many works focus on generalizing and

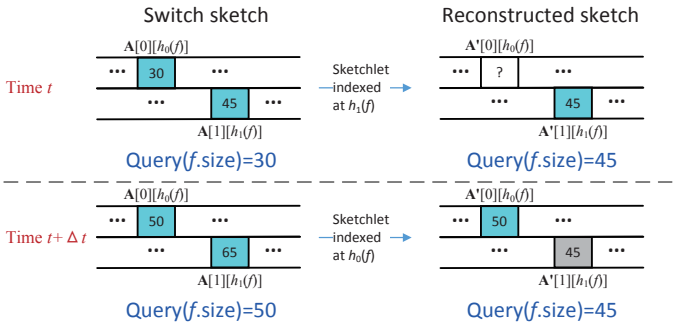


Fig. 1. An example demonstrating the reasons behind inaccuracy of an end-host reconstructed sketch compared with the original switch sketch.

optimizing sketches. Yang et al. [11] present a generic sketch named Elastic sketch that identifies and differentiates large flows from small ones, and is adaptive to traffic variances. To adapt to skewed network flows, Yang et al. [12] propose a novel sketch, namely the Diamond sketch, to dynamically assign appropriate amount of resources for tracing each flow on demand. Liu et al. [13] propose a new sketch called the Slim-Fat (SF) sketch that achieves high accuracy without sacrificing the update and query speed. Song et al. [26] propose a tree-based sketch structure named FCM-sketch as a more accurate and memory-efficient replacement of the count-min sketch. Zhang et al. [14] design a structure named CocoSketch that is capable to support partial key queries. Li et al. [16] present Pyramid Family, a generic framework for flow size estimation with great flexibility. Huang et al. [10], [15] apply compressive sensing to recover measurement results from errors. For optimizing sketch-based systems from a network-wide perspective, UnivMon [9] dispatches measurement tasks to sketches hosted on different switches by solving an integer programming problem; Valerio Bruschi et al. [27] propose to disaggregate a logically centralized sketch into multiple small fragments to distribute the computation overhead.

To combine sketch with INT, Yang et al. [17] place a novel sketch named TowerSketch at network edge to aggregate per-packet INT information. Zhao et al. [18] design a novel sketch named SuMax on switch, and divide it into small-sized sketchlets to send to end-host using INT; on receiving the sketchlets, the end-host reconstructs a sketch that has an identical structure as the one on the switch.

III. Motivation and System Overview

A. Motivation

In this work, we focus on the "reconstructing sketch at end-host" approach for combining sketch with INT, and aim to improve its measurement accuracy. Before presenting our work, we first briefly introduce how sketch and INT are combined, with LightGuardian [18] as an example.

In LightGuardian, a programmable switch maintains two instances of a modified count-min sketch, each instance is composed of w columns and d rows of buckets. One sketch instance is active and updated by incoming packets, and the

other is idle. The two instances swap periodically. A sketchlet in LightGuardian is simply a column of the idle sketch instance containing d buckets as shown in Fig. 2(a). When a switch receives an INT packet, it randomly selects a column as a sketchlet, and embeds it into the packet header to send to end-host. By assembling the received sketchlets, the end-host reconstructs a sketch of an identical structure for answering measurement queries.

In this work, we slightly modify the original LightGuardian design by maintaining only one sketch instance in programmable switch, and transmits its buckets with sketchlets in real-time. Note that our modification actually improves LightGuardian, as in the original design, all the measurement data in the idle sketch is stale. In the rest part of this paper, we use the term LightGuardian to refer to this modified version.

We use an example in Fig. 1 to demonstrate why compared with the original switch sketch, an end-host reconstructed sketch could be inaccurate. Suppose that the sketch is composed of $d = 2$ rows, and it traces flow size in bytes or packets. As shown in the top figure of Fig. 1, at time t, if we estimate the flow f's size with the switch sketch $\mathbf{A}$, the result should be $\min\{\mathbf{A}[0][h_0(f)], \mathbf{A}[1][h_1(f)]\} = 30$. However, suppose that by time t, only the column indexed at $h_1(f)$ has been sent to the end-host by the INT flow, then the bucket $\mathbf{A}'[0][h_0(f)]$ in the reconstructed sketch $\mathbf{A}'$ is invalid, as it does not contain any valid measurement data. If we estimate f's size with $\mathbf{A}'$, the result would be $\min\{\mathbf{A}'[0][h_0(f)], \mathbf{A}'[1][h_1(f)]\} = 45$, which is overestimated, due to the invalid data in $\mathbf{A}'[0][h_0(f)]$.

After Δt seconds, as flow f continues to grow, both $\mathbf{A}[0][h_0(f)]$ and $\mathbf{A}[1][h_1(f)]$ are incremented by 20. Assume that by $t + \Delta t$, the column indexed at $h_0(f)$ has just been sent to the end-host, but the column at $h_1(f)$ has not been sent during $[t, t + \Delta t]$, as demonstrated in the bottom figure of Fig. 1. At $t + \Delta t$, querying flow f with the switch sketch $\mathbf{A}$ returns 50, but querying f with the reconstructed sketch $\mathbf{A}'$ returns 45, which is underestimated, due to the stale data in $\mathbf{A}'[1][h_1(f)]$.

From the example in Fig. 1, one can see that errors arise when the end-host reconstructed sketch contains invalid or stale data in its buckets. Note that in real-world networks, a switch may forward thousands of flows, and each packet updates d buckets in the switch sketch; on the other hand, an INT flow packet carries at most d buckets from the switch to the end-host. Since the INT flow's packet rate is much smaller than the rate of all the other flows combined, buckets of a switch sketch are updated much more frequently than they are synchronized to the end-host. In other words, invalid or stale data in the reconstructed sketch due to intimely synchronization is inevitable. *Given a fixed INT overhead (i.e., INT flow packet rate), our question is: how to smartly select sketch buckets to add to sketchlets, so as to reduce the measurement errors in the end-host reconstructed sketch?*

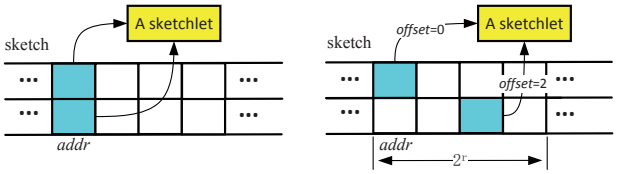
B. System Overview

In this paper, we present *DUNE*, a sketch-INT network measurement system. To overcome the accuracy issue as above discussed, we make two key innovations in DUNE: We first propose a novel sketchlet design named *scatter sketchlet*. Unlike the *column sketchlet* in LightGuardian [18] that contains an entire sketch column, a scatter sketchlet enables a switch to select individual sketch buckets from different columns to add to sketchlets. We further prove in theory that unless the sketch is extremely crowded, a scatter sketchlet has a higher efficiency in transferring measurement data to end-host compared with a column sketchlet.

The second innovation is the methods for selecting sketch buckets to sketchlets. We design a *bitmap* data structure that traces the update status of a sketch bucket, and develop a bitmap-based algorithm to avoid selecting invalid sketch buckets. We also design a counter array structure named *Cookie* for tracing "freshness" of a sketch bucket, and develop a Cookie-based algorithm that selects buckets containing valuable measurement data. Both methods are carefully designed under the stringent constraints of the RMT programmable switch architecture and implemented on a commodity Tofino switch. Table I lists the frequently used notations in this paper.

TABLE I
FREQUENTLY USED NOTATIONS

Denotation	Meaning
$\mathbf{A}$	Switch sketch
$\mathbf{A}'$	End-host reconstructed sketch
$\mathbf{B}$	Bitmap
$\mathbf{C}$	Cookie
d	Num. of sketch/bitmap/Cookie rows
w	Num. of sketch/bitmap/Cookie columns
c	Sketch bucket size in bits
r	Scatter sketchlet's offset length in bits
b	Cookie cell size in bits
N	Num. of network flows traced by a switch sketch



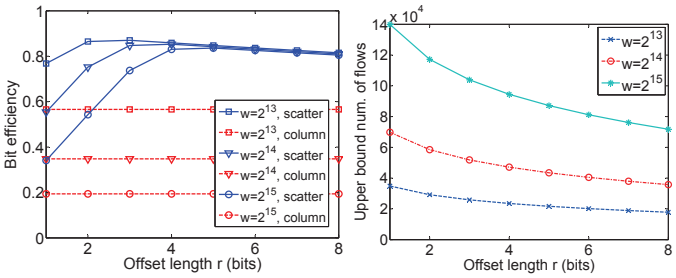
(a) Column sketchlet (b) Scatter sketchlet

Fig. 2. Comparison between column sketchlet and scatter sketchlet.

IV. Scatter Sketchlet

A. Sketchlet Design

Before presenting our sketchlet design, we first describe how a sketch is realized and a sketchlet is formed in a programmable switch (e.g., the Tofino switch). A programmable switch processes packets with a pipeline, which is composed of a series of match-action unit (MAU) stages. Each MAU stage has a stage-local memory, and stateful elements such as sketch buckets are stored as *registers* in the memory. Currently, a register can be accessed at most once by a packet in its pipeline pass, moreover, a register access is limited to one simple read-update-write operation that must be realized in a small piece of code called *register action*. In a sketch-based measurement



(a) Bit efficiency (b) Upper bound number of flows

Fig. 3. (a) Bit efficiencies of scatter and column sketchlets, and (b) upper bound number of flows that a scatter sketchlet can achieve a higher bit efficiency than a column sketchlet. The sketch has $d = 2$ rows and various number w of columns, bucket size is $c = 64$ bits, and the offset length r of the scatter sketchlet varies from 1 to 8 bits.

system, a $d \times w$ sketch is generally realized as d registers, each containing a row of w buckets, and the d registers are placed in d different stage-local memories in switch. When an INT packet enters into the switch's pipeline, it sequentially accesses the d registers, and retrieves the bucket at the specified column index from each register to form a sketchlet.

We propose a novel sketchlet design named *scatter sketchlet*. As shown in Fig. 2, for a $d \times w$ switch sketch $\mathbf{A}$, a scatter sketchlet contains d buckets, one from each row. A scatter sketchlet is addressed as $(addr, offset[1 \cdots d])$, where $addr \in \{1, \cdots, w\}$ is a column index, and $offset[1 \cdots d]$ are d r-bit offsets, where $offset[i]$ indicates the distance between the column index of the bucket in the i^{th} row of the sketch and $addr$. For example, the tuple $(addr, offset[i])$ points to the bucket $\mathbf{A}[i][addr + offset[i]]$. Note that scatter sketchlet doesn't violate the register access restriction, as one register, which implements a sketch row, is still accessed at most once per packet.

On receiving a scatter sketchlet, the end-host updates the corresponding buckets in the reconstructed sketch $\mathbf{A}'$ according to the sketchlet address. After collecting enough sketchlets, the end-host can either use the locally reconstructed sketches to answer measurement queries, or send them to the global analyzing server to perform further analysis.

B. Bit Efficiency

The design of the scatter sketchlet allows its carried bucket to be selected from a range $[addr, (addr + 2^r - 1)]$, which effectively reduces the chance that an invalid bucket is selected. Consider a $w \times d$ sketch tracing a total number of N network flows, a bucket is invalid only when none of the N flows is hashed to it, which happens at a probability of $\left(1 - \frac{1}{w}\right)^N \approx e^{-\frac{N}{w}}$. The probability that there exists at least one valid bucket in the range $[addr, (addr + 2^r - 1)]$ is $\left(1 - e^{-\frac{N}{w} \times 2^r}\right)$, and among the d buckets in a scatter sketchlet, averagely $\left(1 - e^{-\frac{N}{w} \times 2^r}\right) \times d$ of them contain valid measurement data.

Besides the buckets, a sketchlet also needs to carry the addresses of its contained buckets for the end-host to assemble them into the reconstructed sketch. For a column sketchlet, its address is simply the $\log_2 w$-bit column index [18], but for a scatter sketchlet, in addition to the $\log_2 w$-bit $addr$, the address also contains the d r-bit offsets, so the total address size is $(\log_2 w + d \times r)$ bits.

We define *bit efficiency*, which is the ratio between the bits of valid measurement data in a sketchlet and the total sketchlet size, to measure the efficiency of a sketchlet design. From the above analysis, it is easy to see that a scatter sketchlet's bit efficiency is

$$E = \frac{\left(1 - e^{-\frac{N}{w} \times 2^r}\right) \times d \times c}{d \times c + \log_2 w + d \times r} \tag{1}$$

where c is the size of a sketch bucket in bits. Note that a column sketchlet's bit efficiency can be obtained by applying $r = 0$ to (1), as column sketchlet can be viewed as a special case of scatter sketchlet with $r = 0$. Regarding the bit efficiencies of the two sketchlet designs, we have the following result.

Theorem 1. *As long as the number of the network flows N traced by a switch sketch satisfies*

$$N < w \times \ln\left(\frac{d \times c + \log_2 w}{d \times r}\right) \tag{2}$$

a scatter sketchlet with $r \geq 1$ achieves a higher bit efficiency than a column sketchlet.

The proof of Theorem 1 is given in [28].

In Fig. 3(a), we present bit efficiencies of the scatter and column sketchlets as in (1) under various sketch sizes and scatter sketchlet's offset lengths. One can see that the scatter sketchlet has obvious higher bit efficiencies than the column sketchlet under same conditions. In Fig. 3(b), we plot the upper bound of the traced flow numbers in (2). We can see that unless the switch sketch is extremely crowded by tracing much more flows than its sketch buckets, a scatter sketchlet always achieves a higher bit efficiency than a column sketchlet.

V. Bucket Selection Algorithm

The scatter sketchlet allows a switch to pick a sketch bucket in a range of $[addr, (addr + 2^r - 1)]$ to add to sketchlet, however, how to identify the buckets that contain valid and valuable measurement data in the range is still unknown. In this section, we present methods for selecting sketch buckets.

A. Bitmap Algorithm

The first algorithm we propose is called *bitmap algorithm*. As its name suggests, the algorithm maintains within programmable switch a bitmap $\mathbf{B}$, which has same logical structure as the sketch with d rows and w columns of bits. Initially, all bits in $\mathbf{B}$ are set to 0. As presented in Algorithm 1, the bitmap $\mathbf{B}$ is updated on two events:

- When receiving a packet of an ordinary flow f, in addition to updating the sketch buckets $\mathbf{A}[i][h_i(f)]$, the switch also sets all the bits at the same positions in the

Algorithm 1: *Bitmap* algorithm

```
   Input  : A packet of flow f
1  if f is an ordinary flow then
2      for i = 1 ··· d do
3          Update A[i][h_i(f)];
4          B[i][h_i(f)] ← 1;
5  if f is the INT flow then
6      Randomly select addr from {1, ··· , w};
7      for i = 1 ··· d do
8          for j = 0 ··· 2^r − 1 do
9              if B[i][addr + j] == 1 then
10                 break;
11         B[i][addr + j] ← 0;
12         Add A[i][addr + j] to scatter sketchlet;
```

Algorithm 2: *Cookie* algorithm

```
   Input  : A packet of flow f
1  if f is an ordinary flow then
2      for i = 1 ··· d do
3          Update A[i][h_i(f)];
4          C[i][h_i(f)] ← C[i][h_i(f)] + 1;
5  if f is the INT flow then
6      PktCnt + +; Randomly select addr from {1, ··· , w};
7      for i = 1 ··· d do
8          for j = 0 ··· 2^r − 1 do
9              if C[i][addr + j] ≥ (2^h − 1) then
10                 CellCnt + +; break;
11         C[i][addr + j] ← C[i][addr+j]/2;
12         Add A[i][addr + j] to scatter sketchlet;
```

bitmap as 1, i.e., $\mathbf{B}[i][h_i(f)] \leftarrow 1$, for $i = 1, \cdots, d$ (line 1-4).

- When receiving an INT flow packet, the switch randomly selects $addr$ from $\{1, \cdots, w\}$ (line 6), and from each row of the bitmap, it finds the first bit in the range $[addr, (addr + 2^r - 1)]$ whose value is 1, adds the corresponding sketch bucket to the sketchlet, and clears the bit to 0 (line 7-12).

The bitmap algorithm has two properties: First, it avoids selecting invalid bucket into a sketchlet, as an invalid bucket's corresponding bit in the bitmap is always 0; Second, it will not select a bucket if it has not been updated since the last time it was selected into a sketchlet.

B. Cookie Algorithm

Studies show that rate distribution of real-world network flows is highly skewed, and under such a distribution, one flow may grow much faster than another [29], [30]. To cope with such a skewness, we propose an algorithm named *Cookie algorithm*, and present it in Algorithm 2. The algorithm maintains in programmable switch a counter array named *Cookie*, which has same logical structure as the sketch with d rows and w columns of cells. Each Cookie cell is a b-bit counter (where b is significantly smaller than the bucket size c), and all the counters are initialized as 0.

Motivated by TCP's congestion control, the algorithm updates the Cookie cells in an AIMD (additive increase and multiplicative decrease) manner [31]. More specifically, the Cookie $\mathbf{C}$ is updated on two events:

- **Additive increase**: When receiving a packet of an ordinary flow f, in addition to update the sketch buckets $\mathbf{A}[i][h_i(f)]$, the switch also increments the counters $\mathbf{C}[i][h_i(f)]$ by 1, for $i = 1, \cdots, d$ (line 1-4).
- **Multiplicative decrease**: When receiving an INT flow packet, the switch randomly selects $addr$ from $\{1, \cdots, w\}$ (line 6), and for each row in the Cookie, it compares each cell in the range $[addr, (addr + 2^r - 1)]$ against a threshold $(2^h - 1)$ (where $h \leq b$). For the first cell that is no smaller than the threshold, its corresponding sketch bucket is selected into the sketchlet, and the cell's value is halved by right-shifting one bit (line 7-12).

The switch maintains two counters, $CellCnt$ and $PktCnt$. $CellCnt$ records the number of the sketch buckets that have been selected into sketchlets (line 10), and $PktCnt$ is the number of the INT packets the switch has received (line 6). Periodically, the network's measurement control plane computes a ratio $\frac{CellCnt}{d \times PktCnt}$: if the ratio is below a threshold α, the control plane decreases the parameter h by one as $h = h - 1$, which means that the switch will be less selective in selecting buckets into sketchlets; and if the ratio exceeds another threshold β, the switch will behave more selective with $h = h + 1$. For the Cookie algorithm, we have the following result.

Theorem 2. *The frequency of a sketch bucket being selected into sketchlets is statistically proportional to the bucket's update frequency.*

We present the proof of Theorem 2 in [28].

Theorem 2 indicates that the Cookie algorithm is biased towards fast-growing flows, as when a fast-growing flow frequently updates its associated buckets in the switch sketch, the buckets will also be selected into sketchlets frequently. Clearly, this is a desired property, especially when network flows follow a highly skewed rate distribution.

VI. Implementation

A. DUNE Switch Prototype

We have implemented a prototype of the DUNE system on the Edgecore Wedge 100BF Tofino-based programmable switch. For implementing a DUNE switch, two components need to be realized: 1) the sketch structure and the action to access the sketch buckets; 2) the bitmap/Cookie structure and the action to access the bits/Cookie cells based on Algorithm 1 and 2. For realizing the switch sketch, we consult the method in LightGuardian [18] and implement a CM sketch composed of $d = 2$ rows and $w = 2^{15}$ columns of buckets, and the bucket size is $c = 64$ bits. Each sketch row is implemented as

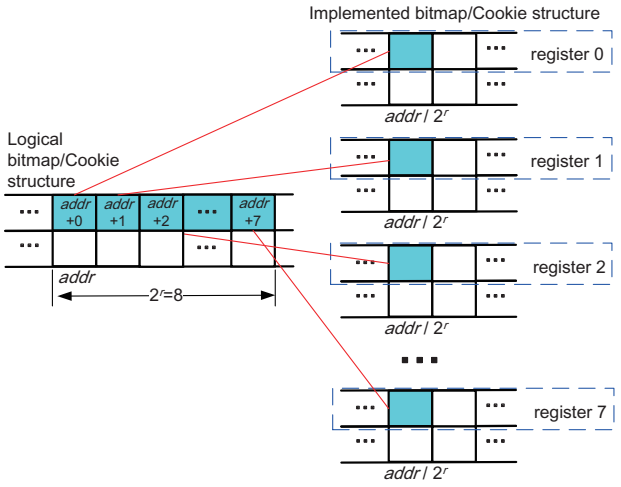


Fig. 4. Implementation of bitmap/Cookie structure on Tofino switch with $r = 3$.

a 256 kB-register, and we realize the operation for updating and retrieving a sketch bucket in one single register action.

Although having same logical structure, however, we can not use the same method to implement the bitmap and Cookie structures, because of the following reason: Recall that in both Algorithm 1 and 2, a sketch bucket is selected from a range of $[addr, (addr + 2^r - 1)]$, and in the worst case, as many as 2^r bits or Cookie cells need to be inspected. If a row of bitmap/Cookie is implemented as one register, under the Tofino switch's register access rule, the operations for inspecting 2^r consecutive bits or Cookie cells must be realized in one single register action. Unfortunately, with the current P4 Tofino switch, it is prohibitive to inspect 2^r consecutive bits or Cookie cells within one single register action.

To overcome this problem, in our implementation, we realize a bitmap/Cookie row with 2^r registers, where each register contains $\frac{w}{2^r}$ bits/Cookie cells. As shown in Fig. 4, a bit/Cookie cell at the position of $[i][k]$ $(i = 1, \cdots, d,\ k = 1, \cdots, \frac{w}{2^r})$ in the j^{th} register corresponds the bit/Cookie cell at the position of $[i][k \cdot 2^r + j]$ in the logical bitmap/Cookie.

For searching in the 2^r consecutive bits/Cookie cells in $[addr, (addr+2^r-1)]$, we sequentially access the bits/Cookie cells indexed at $\frac{addr}{2^r}$ in all the 2^r registers[1]. For a bit/Cookie cell in each register, we check (and update) its value with one single register action. If a bit/Cookie cell indexed at $\frac{addr}{2^r}$ in the j^{th} register is selected, we modify its value and add the bucket $\mathbf{A}[i][addr + j]$ from the sketch to the sketchlet.

In our implementation on the Tofino switch, we set $r = 3$, therefore use 8 registers as in Fig. 4 to implement a bitmap/Cookie row. We have open-sourced our P4 code[2].

B. Overhead Analysis

Commodity programmable switches usually have limited memory capacity, so in the following, we analyze the memory consumptions of the bitmap and Cookie structures. Since logically, bitmap and Cookie have an identical structure as the sketch, their memory consumption are $d \times w$ and $d \times w \times b$ bits respective. In our prototype implementation with $d = 2$, $w = 2^{15}$, and $b = 8$ bits, a bitmap consumes 8 kB, and a Cookie requires 64 kB memory. We can see that neither structure imposes a heavy memory overhead on a programmable switch.

We then analyze the memory access overhead. Since a packet of an ordinary network flow is required to update the sketch buckets as well as the bits/Cookie cells, it accesses $2 \times d = 4$ registers during its pipeline pass; for an INT flow packet, it needs to access $d + d \times 2^r = 18$ registers to select a sketch bucket by inspecting bits/Cookie cells in a range of $[addr, (addr + 2^r - 1)]$. However, we believe that register accesses do not cause a problem, because of two reasons: First, registers in commodity programmable switches are realized with SRAM, which is very fast with a nanosecond-scale access time; Second, the INT flow packets, which make more register accesses, constitute a very small portion of the entire network traffic on the switch. We will evaluate the impact of the register accesses in DUNE on a commodity programmable switch's forwarding performance in Sec. VII-E.

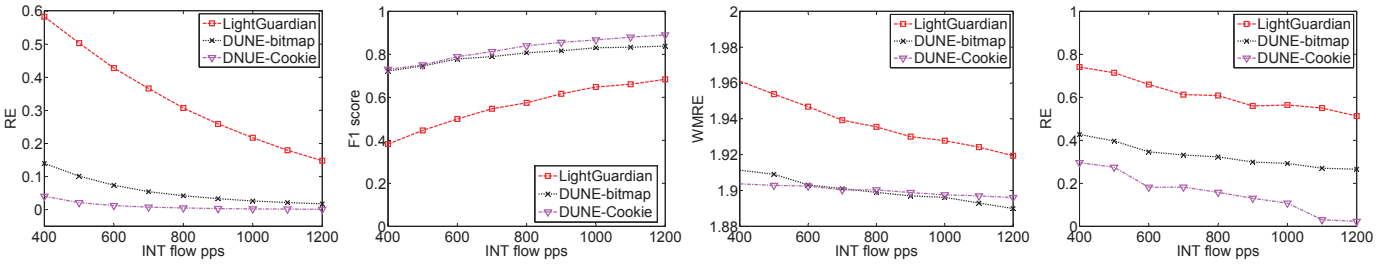
VII. Evaluation

We conduct extensive experiments to evaluate DUNE, and in particular, we examine the following three sketch-INT network measurement systems.

- **DUNE-bitmap**: In DUNE-bitmap, we employ the scatter sketchlet as described in Sec. IV, and use the bitmap algorithm in Algorithm 1 to select sketch buckets to send to end-host.
- **DUNE-Cookie**: The DUNE-Cookie system employs the scatter sketchlet and applies the Cookie algorithm in Algorithm 2 to add sketch buckets to sketchlets.
- **LightGuardian**: As a representative sketch-INT system, LightGuardian [18] adopts the column sketchlet, and employs an algorithm named k+*chance* to select sketch columns. In k+*chance*, a programmable sketch maintains k bit arrays, each containing w bits corresponding to the w sketch columns. When a column index $addr$ is randomly selected, the switch sequentially inspects the bits indexed at $addr$ in each array: If a 0 is encountered, the switch sets the bit as 1, and adds the sketch column at $addr$ to the sketchlet; If all the bits have already been set to 1, the switch randomly selects an other column as the sketchlet. Ideally with k+*chance*, all the sketch columns will be sent to end-host with a fair chance.

We have implemented DUNE-bitmap and DUNE-Cookie with Tofino switches. We also implement DUNE-bitmap, DUNE-Cookie, and LightGuardian on `bmv2` [32], which is a P4-programmable software switch. Unless otherwise specified, in the following experiments, we set the sketch/bitmap/Cookie size as $d = 2$ rows and $w = 2^{15}$ columns. A sketch bucket contains $c = 64$ bits, the size of a Cookie cell is $b = 8$ bits, and the length of a scatter sketchlet's offset is $r = 3$ bits. For the

[1]In our implementation, $addr$ is randomly selected as multiples of 2^r.
[2]https://github.com/DuneHPCC724/Dune



(a) Cardinality (b) Heavy hitter (c) Flow size distribution (d) Entropy

Fig. 5. Accuracies of (a) cardinality estimations in RE, (b) heavy hitter detections in F1-score, (c) flow size distribution estimations in WMRE, and (d) entropy estimations in RE with LightGuardian, DUNE-bitmap, and DUNE-Cookie under various INT flow pps.

DUNE-Cookie system, we set the two threshold parameters as $\alpha = 0.5$ and $\beta = 1.0$. For evaluating LightGuardian, we employ $k = 8$ bit arrays as suggested in [18], which means that the bit arrays used by the k+*chance* algorithm consume four times memory compared with DUNE-bitmap, or half of the memory compared with DUNE-Cookie.

With the above parameter settings, the size of a column sketchlet is $d \times c + \log_2 w = 143$ bits, and the size of a scatter sketchlet is $d \times c + \log_2 w + d \times r = 149$ bits. We can see that DUNE is still lightweight by increasing LightGuardian's INT overhead no more than 4.2%.

We use the public available MAWI packet trace [33] captured from the WIDE backbone to drive the experiments. The trace contains 9.6M flows, and we randomly select $6,000$ flows from the top-50K largest flows for each experiment.

A. Measurement Accuracy

1) Measurement tasks and metrics: We first perform a number of network measurement tasks with the three sketch-INT systems. The tasks are:

- **Cardinality estimation**. In this task, we count number of the distinct flows to estimate the traffic cardinality.
- **Heavy hitter detection**. This task aims to identify the top 10% largest flows in the traffic[3].
- **Flow size distribution estimation**. This task aims to estimate m_i, the number of the flows of size i for all the possible sizes.
- **Entropy estimation**. This task estimates the entropy of the traffic, which is defined as

$$Entropy = \sum_i \left(i \times \frac{m_i}{M} \times \log \frac{m_i}{M} \right)$$

where m_i is the flows of size i and $M = \sum_i m_i$.

Note that all the above estimations are conducted on the sketches reconstructed by different sketch-INT systems on end-hosts, rather than on the original switch sketches. We use the following metrics to evaluate the measurement accuracies.

[3]Neither LightGuardian nor DUNE transmit flow IDs, so in our experiment, we assume that end-host already has the knowledge of flow IDs obtained from side channels such as server logs.

- **Relative error (RE)**: We use the relative error, which is defined as

$$RE = \frac{|Estimated - Truth|}{Truth}$$

to evaluate the cardinality and entropy estimations' accuracies.
- **F1-score**. For detecting heavy hitters, we use the F1-score to evaluate the estimation accuracy.
- **Weighted mean relative error (WMRE)**: We compare the estimated flow size distribution with the ground truth, and compute WMRE as

$$WMRE = \frac{\sum_i |m_i - \hat{m}_i|}{\sum_i \left(\frac{m_i + \hat{m}_i}{2} \right)}$$

where m_i and $\hat{m}_i$ are the estimated and ground-truth numbers of the flows of size i.

2) Results: We conduct the measurement tasks with the three sketch-INT systems, and present the results in Fig. 5. In each experiment, we vary the INT flow's packets-per-second (pps), which decides the number of the sketchlets that can be transferred from the switch sketch to the end-host, from 400 to $1,200$ in the experiments.

Several interesting observations can be made from Fig. 5. The first observation is that all the Sketch-INT systems have better performances with the increase of the INT flow's pps. This is easy to understand, as a higher pps indicates that the INT flow can bring more buckets to the end-host for reconstructing the sketch.

The second observation is that our proposed systems, i.e., DUNE-bitmap and DUNE-Cookie, are more accurate in all the tasks than LightGuardian. This is because compared with LightGuardian, our proposed scatter sketchlet and bucket selection algorithms enable the DUNE systems to transfer more valuable measurement data to end-hosts, despite that in the three systems, end-hosts receive same amount of sketchlets.

The third observation is that DUNE-Cookie outperforms DUNE-bitmap in most cases. This is because DUNE-Cookie has the desired property of selecting sketch buckets at frequencies that are proportional to the buckets' update frequencies, as stated in Theorem 2. The only exception is the flow size distribution estimations in Fig. 5(c), in which DUNE-bitmap outperforms DUNE-Cookie when the INT flow pps is small.

The reason behind is that DUNE-bitmap uniformly selects sketch buckets of all flows; while DUNE-Cookie is biased towards the fast-growing flows, and when there are relatively fewer INT packets, small flows are ignored, which leads to a distorted estimation on the flow size distribution.

B. Decomposing Measurement Inaccuracy

The evaluation results in Fig. 5 suggest that errors pervasively exist in the end-host reconstructed sketch. Errors may come from two different sources: First, errors could be caused by hash collisions in the switch sketch, and the erroneous measurement data is transferred to the end-host in sketchlets. Second, the measurement data in the switch sketch is error-free, but the reconstructed sketch at the end-host is not timely synchronized with the switch sketch, thus is inaccurate because of the invalid or stale data in the sketch buckets, as we have seen in Sec. III-A. In the following, we seek to identify and quantify the two types of the errors.

1) Decomposing methods and metrics: Before presenting the methods and metrics for decomposing the errors, we first introduce some notations. Let $\mathbf{x} = \{x_f | f \in \mathbb{F}\}$ be a set of measurement data on a network state (i.e., flow size) over a flow set $\mathbb{F}$, and $\mathbf{y} = \{y_f | f \in \mathbb{F}\}$ be another set of measurement data on the same state over a same flow set $\mathbb{F}$. We define the *Relative Aggregated Error (RAE)* for comparing the measurement data $\mathbf{y}$ against $\mathbf{x}$ as

$$RAE(\mathbf{y}, \mathbf{x}|\mathbb{F}) = \frac{\sum_{f\in\mathbb{F}} |y_f - x_f|}{\sum_{f\in\mathbb{F}} x_f} \tag{3}$$

We use the following metrics to quantify the errors from different sources.

- $RAE(\mathbf{n_A}, \mathbf{n}|\mathbb{F})$: It is the RAE for comparing the measurement data in the switch sketch against the ground truth, where $\mathbf{n_A} = \{n_f^{\mathbf{A}} | f \in \mathbb{F}\}$ is the flow sizes estimated with the switch sketch $\mathbf{A}$, and $\mathbf{n} = \{n_f | f \in \mathbb{F}\}$ is the set of the ground-truth flow sizes.
- $RAE(\mathbf{n_{A'}}, \mathbf{n}|\mathbb{F})$: It is the RAE for comparing the measurement data in the end-host reconstructed sketch against the ground truth, where $\mathbf{n_{A'}} = \{n_f^{\mathbf{A}'} | f \in \mathbb{F}\}$ is the flow sizes estimated with the reconstructed sketch $\mathbf{A}'$.
- $RAE(\mathbf{n_{A'}}, \mathbf{n_A}|\mathbb{F})$: It is the RAE for comparing the flow sizes estimated with the end-host reconstructed sketch $\mathbf{A}'$ against the ones estimated with the switch sketch $\mathbf{A}$.

From the above definition, we can see that $RAE(\mathbf{n_A}, \mathbf{n}|\mathbb{F})$ quantifies the errors caused by hash collisions in the switch sketch, $RAE(\mathbf{n_{A'}}, \mathbf{n_A}|\mathbb{F})$ measures the errors caused by the invalid and stale data in the end-host reconstructed sketch, and $RAE(\mathbf{n_{A'}}, \mathbf{n}|\mathbb{F})$ captures the overall errors.

2) Results: We run the three sketch-INT systems to estimate the sizes of the flows from the MAWI trace, and compare RAEs of the different systems in Fig. 6. In particular, we compare the flow sizes estimated with the switch sketch against the ground truth, and present $RAE(\mathbf{n_A}, \mathbf{n}|\mathbb{F})$ denoted as "switch sketch" in Fig. 6(a); we also compare the end-host sketches reconstructed by different systems against the ground truth in $RAE(\mathbf{n_{A'}}, \mathbf{n}|\mathbb{F})$ in the figure. In Fig. 6(b),

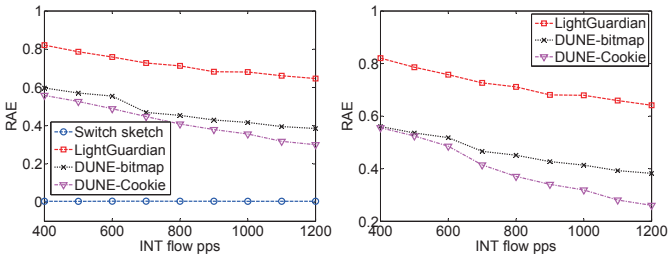


Fig. 6. (a) $RAE(\mathbf{n_A}, \mathbf{n}|\mathbb{F})$ of switch sketch and $RAE(\mathbf{n_{A'}}, \mathbf{n}|\mathbb{F})$ of end-host reconstructed sketches against ground truths by LightGuardian, DUNE-bitmap, and DUNE-Cookie; (b) $RAE(\mathbf{n_{A'}}, \mathbf{n_A}|\mathbb{F})$ of end-host reconstructed sketches against switch sketches by LightGuardian, DUNE-bitmap, and DUNE-Cookie under various INT flow pps.

we compare the end-host reconstructed sketches against the switch sketches, and present $RAE(\mathbf{n_{A'}}, \mathbf{n_A}|\mathbb{F})$ of the three systems.

From Fig. 6, we can make the following observations. First, the reconstructed sketches at end-hosts contain much more errors compared with the switch sketch, suggesting that most of the measurement errors are caused by the invalid and stale data in the reconstructed sketch. Second, a high INT flow packet rate can considerably reduce the errors in the reconstructed sketches, and the DUNE systems benefit more from a higher INT rate. For example, when the INT flow pps is increased from 400 to $1,200$, $RAE(\mathbf{n_{A'}}, \mathbf{n}|\mathbb{F})$ are reduced by 21.8%, 31.8%, and 53.1% for LightGuardian, DUNE-bitmap, and DUNE-Cookie respectively. Finally, thanks to the scatter sketchlet and the smart bucket selection algorithms, our proposed DUNE systems achieve much lower error rates than LightGuardian. For example, under the $1,200$ INT flow pps, DUNE-bitmap and DUNE-Cookie reduce LightGuardian's $RAE(\mathbf{n_{A'}}, \mathbf{n}|\mathbb{F})$ by as much as 40.4% and 59.4% respectively.

C. Impact of Offset Length r

The design of the scatter sketchlet enables a switch to select buckets in a range of $[addr, (addr + 2^r - 1)]$. Intuitively, the larger the offset length r is, with a larger chance a sketch bucket containing "fresh" measurement data could be selected, and the higher estimation accuracy the reconstructed sketch will achieve.

In this experiment, we run different sketch-INT systems under various offset length r ranging from 2 to 10 bits, and present $RAE(\mathbf{n_{A'}}, \mathbf{n_A}|\mathbb{F})$ of LightGuardian, DUNE-bitmap, and DUNE-Cookie in Fig. 7. We also plot $RAE(\mathbf{n_A}, \mathbf{n}|\mathbb{F})$ for comparison. From the figure we can see that increasing the offset length do reduce the errors in the DUNE systems, but the reduction is not very significant. For example, for DUNE-Cookie, by increasing r from 2 to 10 bits, the error reduction is only 5.6%. Recall that in our Tofino implementation, a packet accesses up to 2^r registers for selecting a sketch bucket. The result in Fig. 7 suggests that with a small offset length (e.g., $r = 3$), the systems of DUNE-bitmap and DUNE-Cookie can achieve decent accuracies.

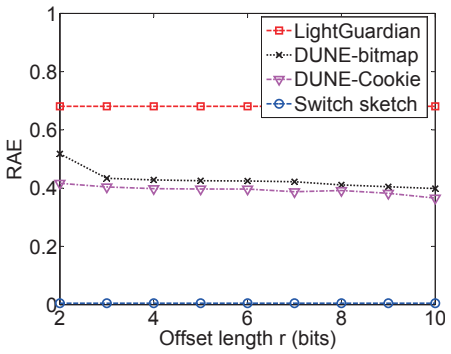


Fig. 7. $RAE(\mathbf{n}_{\mathbf{A}'}, \mathbf{n}_{\mathbf{A}}|\mathbb{F})$ of reconstructed sketches against switch sketches by LightGuardian, DUNE-bitmap, and DUNE-Cookie, and $RAE(\mathbf{n}_{\mathbf{A}}, \mathbf{n}|\mathbb{F})$ of switch sketch against ground truth under various offset length r.

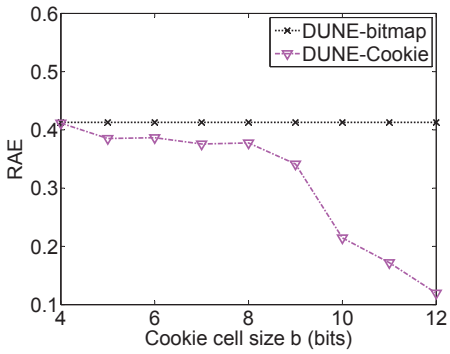


Fig. 8. $RAE(\mathbf{n}_{\mathbf{A}'}, \mathbf{n}_{\mathbf{A}}|\mathbb{F})$ of end-host reconstructed sketches by DUNE-bitmap and DUNE-Cookie under various Cookie cell size b.

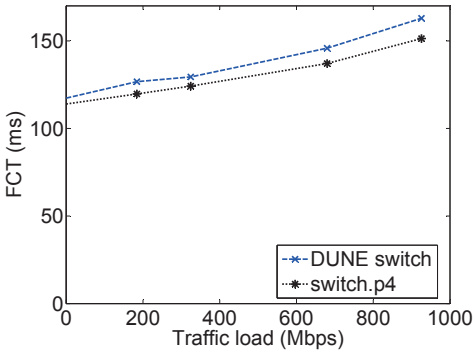


Fig. 9. Comparison of FCTs of a $50,000$-packet INT flow forwarded by DUNE switch and FCTs of a same-sized flow forwarded by baseline switch under various background traffic load.

D. Impact of Cookie Cell Size b

In the DUNE-Cookie system, we employ a Cookie data structure, which logically has an identical structure as the switch sketch, to trace "freshness" of the measurement data in sketch buckets. In this experiment, we investigate the impact of the Cookie cell size by varying b from 4 to 12 bits, and present $RAE(\mathbf{n}_{\mathbf{A}'}, \mathbf{n}_{\mathbf{A}}|\mathbb{F})$ of the DUNE-Cookie system in Fig. 8. We also plot DUNE-bitmap's $RAE(\mathbf{n}_{\mathbf{A}'}, \mathbf{n}_{\mathbf{A}}|\mathbb{F})$ for comparison.

From Fig. 8, we can see that by increasing the Cookie cell size, better accuracies can be achieved by DUNE-Cookie. In particular, when b exceeds 9 bits, the errors are considerably reduced. This is because with the MAWI traffic trace, which has a skewness factor around 1.4, most fast-growing flows can be accurately identified by the Cookie algorithm when a Cookie cell is capable to trace up to $2^9 = 512$ updates. Our observation suggests that a tradeoff is allowed between memory consumption and measurement accuracy: For hardware switches that lack memory resources, a small Cookie cell size can ensure a reasonable accuracy, while in software switches like OVS [34], we can pursue a higher accuracy at a cost of a larger memory usage.

E. Forwarding Performance

As analyzed in Sec. VI, in a DUNE switch, a regular flow's packet accesses $2 \times d$ registers, and an INT flow packet accesses $d + d \times 2^r$ registers respectively during their pipeline passes. In this section, we evaluate the impact of the register accesses to a switch's packet forwarding performance.

We apply the parameters in Sec. VI to implement a DUNE switch on an Edgecore Wedge 100BF Tofino-based programmable switch, and set up a simple testbed composed of two end-hosts and one Tofino-based DUNE switch, where the switch forwards packets sent from the source host to the destination. To evaluate the DUNE switch's forwarding performance, we send an INT flow composed of $50,000$ packets back-to-back to the switch, and measure the flow's completion time (FCT) between the moments when the first packet is sent out and the last packet arrives the destination. In addition to the INT flow, we also send a background traffic varying from 0 to $950\,\mathrm{Mbit/s}$ to the switch. Note that packets from both the INT flow and the background traffic access registers when they enter into the switch's pipeline.

To make a comparison, we run `switch.p4` [35], a baseline L2/L3 switch implementation on another Tofino switch. We send the same background traffic to the switch, and measure a $50,000$-packet flow's FCT. Note that in the baseline switch, a packet does not access any registers. We present the FCTs of the INT flow forwarded by the DUNE switch as well as the FCTs of the 50,000-packet flow forwarded by the baseline switch in Fig. 9. We can see that the INT flow has an FCT slightly longer than the baseline due to register accesses: When there is no background traffic load, the DUNE switch prolongs the FCT by 2.93%; and under the $950\,\mathrm{Mbit/s}$ background traffic load, the FCT is prolonged by 7.63%. The experiment results suggest that register accesses introduced by DUNE only slightly impact a commodity programmable switch's forwarding performance, and a DUNE switch is practical to handle real-world network traffic.

VIII. Conclusion

In this paper, we presented *DUNE*, a lightweight and accurate sketch-INT network measurement system. We revealed the errors caused by the invalid and stale measurement data in the end-host reconstructed sketch, and to combat the errors, we made two key innovations: First, we designed a novel *scatter sketchlet* that allows a switch to select individual buckets to add to sketchlets; Second, we developed data structures for tracing "freshness" of sketch buckets, and proposed algorithms for smartly selecting buckets to send to end-host. We theoretically proved that our proposed methods have higher efficiency in transferring measurement data, and better adapt to skewed network traffic. We implemented DUNE on a commodity Tofino-based switch under the device's stringent constraints. We extensively evaluated our proposed system with experiments driven by real-world traffic, and showed that DUNE can significantly improve the measurement accuracy, at a trivial cost on switch's forwarding performance.

REFERENCES

[1] V. Jeyakumar, M. Alizadeh, Y. Geng, C. Kim, and D. Mazières, "Millions of little minions: Using packets for low latency network programming and visibility," in *Proc. SIGCOMM'14*, Chicago, IL, USA, Aug. 2014.

[2] C. Kim, A. Sivaraman, N. Katta, A. Bas, A. Dixit, and L. J. Wobker, "Inband network telemetry via programmable dataplanes," in *Proc. SIGCOMM'15*, London, UK, Aug. 2015.

[3] P. Tammana, R. Agarwal, and M. Lee, "Simplifying datacenter network debugging with pathdump," in *Proc. OSDI'16*, Savannah, GA, USA, Nov. 2016.

[4] S. Sheng, Q. Huang, and P. P. C. Lee, "DeltaINT: Toward general in-band network telemetry with extremely low bandwidth overhead," in *Proc. ICNP'21*, Dallas, TX, USA, Nov. 2021.

[5] E. Song, T. Pan, C. Jia, W. Cao, J. Zhang, T. Huang, and Y. Liu, "INT-label: Lightweight in-band network-wide telemetry via interval-based distributed labelling," in *Proc. INFOCOM'21*, Vancouver, BC, Canada, May 2021.

[6] R. B. Basat, S. Ramanathan, Y. Li, G. Antichi, M. Yu, and M. Mitzenmacher, "PINT: Probabilistic in-band network telemetry," in *Proc. SIGCOMM'20*, Virtual Event, NY, USA, Aug. 2020.

[7] W. Chen, Y. Tian, Z. Wei, J. Pan, and X. Zhang, "Task scheduling for probabilistic in-band network telemetry," *IEEE/ACM Trans. Netw.*, vol. 30, no. 6, 2022.

[8] Y. Li, R. Miao, C. Kim, and M. Yu, "FlowRadar: A better NetFlow for data centers," in *Proc. NSDI'16*, Santa Clara, CA, USA, Mar. 2016.

[9] 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. SIGCOMM'16*, Florianopolis, Brazil, Aug. 2016.

[10] Q. Huang, X. Jin, P. P. C. Lee, R. Li, L. Tang, Y.-C. Chen, and G. Zhang, "SketchVisor: Robust network measurement for software packet processing," in *Proc. SIGCOMM'17*, Los Angeles, CA, USA, Aug. 2017.

[11] T. Yang, J. Jiang, P. Liu, Q. Huang, J. Gong, Y. Zhou, R. Miao, X. Li, and S. Uhlig, "Elastic Sketch: Adaptive and fast network-wide measurements," in *Proc. SIGCOMM'18*, Budapest, Hungary, Aug. 2018.

[12] T. Yang, S. Gao, Z. Sun, Y. Wang, Y. Shen, and X. Li, "Diamond sketch: Accurate per-flow measurement for big streaming data," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 12, 2019.

[13] L. Liu, Y. Shen, Y. Yan, T. Yang, M. Shahzad, B. Cui, and G. Xie, "SF-Sketch: A two-stage sketch for data streams," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 10, 2020.

[14] Y. Zhang, Z. Liu, R. Wang, T. Yang, J. Li, R. Miao, P. Liu, R. Zhang, and J. Jiang, "CocoSketch: High-performance sketch-based measurement over arbitrary partial key query," in *Proc. SIGCOMM'21*, Virtual Event, USA, Aug. 2021.

[15] Q. Huang, S. Sheng, X. Chen, Y. Bao, R. Zhang, Y. Xu, and G. Zhang, "Toward nearly-zero-error sketching via compressive sensing," in *Proc. NSDI'21*, Apr. 2021.

[16] Y. Li, X. Yu, Y. Yang, Y. Zhou, T. Yang, Z. Ma, and S. Chen, "Pyramid Family: Generic frameworks for accurate and fast flow size measurement," *IEEE/ACM Trans. Netw.*, vol. 30, no. 2, 2022.

[17] K. Yang, Y. Li, Z. Liu, T. Yang, Y. Zhou, J. He, J. Xue, T. Zhao, Z. Jia, and Y. Yang, "SketchINT: Empowering INT with TowerSketch for per-flow per-switch measurement," in *Proc. ICNP'21*, Dallas, TX, USA, Nov. 2021.

[18] Y. Zhao, K. Yang, Z. Liu, T. Yang, L. Chen, S. Liu, N. Zheng, R. Wang, H. Wu, Y. Wang, and N. Zhang, "LightGuardian: A full-visibility, lightweight, in-band telemetry system using sketchlets," in *Proc. NSDI'21*, Apr. 2021.

[19] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN," in *Proc. SIGCOMM'13*, Hong Kong, China, Aug. 2013.

[20] S. Chole, A. Fingerhut, S. Ma, A. Sivaraman, S. Vargaftik, A. Berger, G. Mendelson, M. Alizadeh, S.-T. Chuang, I. Keslassy, A. Orda, and T. Edsall1, "dRMT: Disaggregated programmable switching," in *Proc. SIGCOMM'17*, Los Angeles, CA, USA, Aug. 2017.

[21] S. Tang, S. Zhao, X. Pan, and Z. Zhu, "How to use in-band network telemetry wisely: Network-wise orchestration of Sel-INT," *IEEE/ACM Trans. Netw.*, 2022, doi: 10.1109/TNET.2022.3194086.

[22] C. Estan, G. Varghese, and M. Fisk, "Bitmap algorithms for counting active flows on high speed links," *IEEE/ACM Tans. Networking*, vol. 14, no. 5, 2006.

[23] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese, "An improved construction for counting bloom filters," in *Proc. European Symposium on Algorithms (ESA'06)*, Zurich, Switzerland, Sep. 2006.

[24] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *Journal of Algorithms*, vol. 55, no. 1, 2005.

[25] H. Song, S. Dharmapurikar, J. Turner, and J. Lockwood, "Fast hash table lookup using extended bloom filter: an aid to network processing," *ACM SIGCOMM CCR*, vol. 35, no. 4, 2005.

[26] C. H. Song, P. G. Kannan, B. K. H. Low, and M. C. Chan, "FCM-Sketch: Generic network measurements with data plane support," in *Proc. CoNEXT'20*, Barcelona, Spain, Dec. 2020.

[27] V. Bruschi, R. B. Basat, Z. Liu, G. Antichi, G. Bianchi, and M. Mitzenmacher, "DISCOvering the heavy hitters with disaggregated sketches," in *Proc. CoNEXT'20, poster*, Barcelona, Spain, Dec. 2020.

[28] Z. Wei, Y. Tian, W. Chen, L. Gu, and X. Zhang, "DUNE: Improving accuracy for sketch-INT network measurement systems," 2022, arXiv:2212.04816 [cs.NI].

[29] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. IMC'10*, Melbourne, Australia, Nov. 2010.

[30] Y. Chen, S. Jain, V. K. Adhikari, Z.-L. Zhang, and K. Xu, "A first look at inter-data center traffic characteristics via yahoo! datasets," in *Proc. IEEE INFOCOM'11*, Shanghai, China, Apr. 2011.

[31] D.-M. Chiu and R. Jain, "Analysis of increase and decrease algorithms for congestion avoidance in computer networks," *Comput. Netw. ISDN Syst.*, vol. 17, no. 1, 1989.

[32] "bmv2, the behavioral model for P4," accessed on Mar. 5, 2022. [Online]. Available: https://github.com/p4lang/behavioral-model

[33] "MAWI working group traffic archive," accessed on Mar. 5, 2022. [Online]. Available: https://mawi.wide.ad.jp/mawi/

[34] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. S. Jesse Gross, Alex Wang, P. Shelar, K. Amidon, and M. Casado, "The design and implementation of open vSwitch," in *Proc. NSDI'15*, Oakland, CA, USA, May 2015.

[35] "Consolidated switch repo," accessed on Mar. 5, 2022. [Online]. Available: https://github.com/p4lang/switch