Telemetry Retrieval Inaccuracy in Programmable Switches: Analysis and Recommendations

Hun Namkung[★], Daehyeok Kim[★], Zaoxing Liu[†], Vyas Sekar[★], Peter Steenkiste[★]
[★]Carnegie Mellon University, [†]Boston University

Abstract

Sketching algorithms or sketches are attractive as telemetry capabilities on programmable hardware switches since they offer rigorous accuracy guarantees and use compact data structures. However, we find that in practice, their actual implementations can have a significant (up to 94×) accuracy drop compared to theoretical expectations. We find that the delays incurred by pulling and resetting the data plane state induce accuracy degradation. We design and implement solutions to reduce the delays and show that our solutions can help eliminate almost all the inaccuracy of existing sketch workflows.

CCS Concepts

• **Networks** → **Programmable networks**; **Network monitoring**.

ACM Reference Format:
Hun Namkung, Daehyeok Kim, Zaoxing Liu, Vyas Sekar, Peter Steenkiste . 2021. Telemetry Retrieval Inaccuracy in Programmable Switches: Analysis and Recommendations. In *The ACM SIGCOMM Symposium on SDN Research (SOSR) (SOSR '21), October 11–12, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3482898.3483357

1 Introduction

Recent advances have made it possible to design and implement various telemetry capabilities, such as sketches [15, 18, 19], counting bloom filters [20], and others [17] in programmable switches [1, 3]. At a high level, these network telemetry tasks maintain data structures with arrays of counters in the data plane for tracking traffic flows, which are then retrieved by the switch and network control plane.

Our specific focus in this work is on sketches. The typical workflow of sketch-based telemetry is as follows. For every (pre-defined) measurement *epoch* (i.e., a periodic time window), the switch control plane fetches the counter arrays to compute statistics of interest (e.g., heavy hitters, distinct flows, entropy [10, 15, 18, 19, 26, 31, 39]) and resets the counter arrays. Essentially, the counter arrays are *shared state* between the data plane and the control plane. The data plane updates the state when processing packets, and the control plane reads the state per epoch and resets the state for the next epoch.

While the fidelity of the sketches is backed by theoretical analysis [15, 18, 19], in practice when we implement and deploy sketches using the above workflow on programmable hardware switches (e.g., Intel Tofino-based switch), the empirical results are inaccurate (§2). For instance, there is a significant accuracy drop (e.g., up to 94× error increase), when the epoch size is small (e.g., 5s to 1s). To the best of our knowledge, we are the first to document this counter retrieval problem and propose solutions.

We systematically investigate the state fetching and resetting process implemented on an Intel Tofino-based switch [1]. Our analysis shows that the time spent on pulling and resetting data plane states is non-trivial. We decompose delays into the control and data plane delays, identify a total of six potential delays, and quantify the impact of each component. Our analysis reveals that two control plane delays can cause significant impacts on the accuracy of counters (§3).

Having identified the key bottlenecks, we propose four correct-by-construction solution building blocks, within the scope of sketching algorithms, with different trade-offs:

- Duplicating sketch instances in the data plane, one of which is updated alternately in successive epochs.
- For sketches with a linearity property [20, 26, 30, 35], the control plane can offset the error by subtracting counter arrays between previous and current epoch.
- Deferring a control plane read operation after a reset operation to reduce the impact of the bottleneck delay.
- Using a faster bulk reset API.

We also propose guidelines on which building blocks are appropriate for different use cases (§4). We implement these building blocks for five sketches [15, 18, 19, 21, 31] and evaluate them on a Tofino-based programmable switch [1]. We demonstrate that delays are reduced by more than 95%, and error is reduced by more than 97% (§5). While our focus is on sketches, our findings and solutions are likely more broadly applicable to other telemetry tasks since they often share workflow interactions between the data and control plane.

2 Background and Motivation

We first describe the common workflow of sketches in Fig. 1. We then highlight the sketch accuracy drop in an actual hardware implementation compared to a software implementation and discuss the implications of this observation.

Typical Workflow. Fig. 1 shows the common workflow for deploying network telemetry tasks on programmable switches. Traffic is chunked into time intervals or *epochs*. On the data plane, network telemetry tasks maintain counter arrays that are updated by

SOSR '21, October 11–12, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-9084-2/21/10…$15.00
https://doi.org/10.1145/3482898.3483357

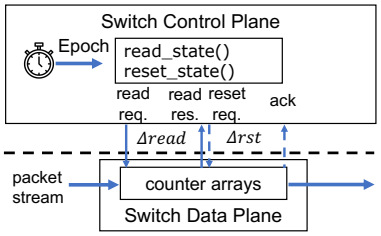
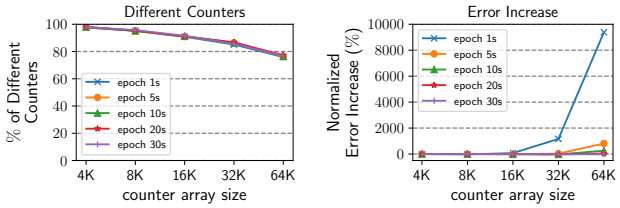


Figure 1: Workflow of sketches.



(a) Different Counters. **(b) Error Increase.**

Figure 2: Different counters cause accuracy degradation.

processing packets. At the end of every epoch, the control plane periodically reads the counter arrays and resets them.

We implement five published sketches [15, 18, 19, 21, 31] on a Tofino-based programmable switch using the above workflow and observed a significant discrepancy in accuracy compared with a software implementation. We illustrate the problem using a simple sketch called count-min sketch (CM) [18]. The CM uses a 2D array of counters to detect heavy hitters from a packet stream for a given flowkey (e.g., srcIP). We use srcIP as the flowkey for our implementation.

Methodology. The sketch implementation on the hardware is partitioned across the data and control plane. In the data plane, we run the CM written in P4 [13] and send the input packet stream (S) to the switch from a directly connected server using `tcpreplay` [6]. The control plane periodically reads and resets the counter arrays using the control plane API provided by the Tofino SDK [2]. The SDK supports Python and C++, and we present results using C++ API.[1] To obtain the theoretically expected accuracy, we use a software implementation of CM sketch written in C++. We split an input packet stream S into multiple subsets S_i corresponding to an epoch with length L.[2] The software implementation pauses packet processing while it reads and resets the counter array. We measured the accuracy of the sketch using both the software and hardware implementations for different epoch lengths (L) and counter array sizes.

Findings. Fig. 2a shows the percentage of the number of different counters in the counter array between the software simulation and the hardware measurement. We see that up to 98% of counters are different. This discrepancy problem reduces accuracy as shown in Fig. 2b. The normalized error increase is defined as $\frac{Error_{actual} - Error_{exp}}{Error_{exp}}$, where $Error_{exp}$ is the expected error using software sketch implementation relative to ground truth and $Error_{actual}$ is the actual error using the hardware implementation. The error increases up to 94× at an array size of 64K and epoch length of 1. We use an average relative error (§5.1) as the error of CM.

Implications. At a high level, the discrepancy arises due to the delays involved in the read and reset operation in Fig. 1, $\Delta read$ and Δrst. As we will see later (§3), they are not negligible. Note that the above results focus on a simple sketch with a small counter array, a relatively large epoch (1 to 30 seconds), and non-adversarial traffic conditions. In practice, the problem could be worse.

- First, richer network telemetry tasks that use more data plane counters such as R-HHH [10] and UnivMon [31] will be impacted more as the impact increases with the size of the counter array.
- Second, network telemetry tasks with tighter timing deadlines (shorter epochs) will be impacted more as the delay, relative to the epoch length, becomes more significant.
- Lastly, the worst case error can become unbounded; there can be bursts of packets (e.g., anomalies or attacks) that coincide with the $\Delta read$ or Δrst intervals.

3 Problem Diagnosis

We take a closer look at the read and reset delays to understand the discrepancy problem better.

3.1 A Closer Look at Sources of Error

We can logically decompose the read and reset delays into control and data plane delays, as shown in Fig. 3. The read delay at $Epoch_i$, $\Delta read_i$, consists of two control plane delays and one data plane delay: $\Delta read_i = \Delta read_i^{C1} + \Delta read_i^{C2} + \Delta read_i^{D}$, where $\Delta read_i^{D}$ represents the duration of actual data transfer from the data plane. Similarly, we can represent the the reset delay as $\Delta rst_i = \Delta rst_i^{C1} + \Delta rst_i^{C2} + \Delta rst_i^{D}$, with similar control and data plane components.

Let $F(S)$ be the sketching function computed on a given set of packets S. For $Epoch_i$, we want to measure $F(S_i)$. However, the above delays result in a different set of packets actually being monitored; they are marked as epoch packets and measured packets in Fig. 4.

More specifically (see Fig. 3), let $S_{\Delta read_i}$ and $S_{\Delta rst_i}$ denote the sets of packet streams during the read and reset operation in $Epoch_i$. Let $S_{\Delta read_i^{C1}}$, $S_{\Delta rst_i^{C1}}$ denote the sets of packets during $\Delta read_i^{C1}$ and Δrst_i^{C1}. $S_{\Delta read_i^{D}}$, $S_{\Delta rst_i^{D}}$ are more subtle because the control plane and data plane access the counter array simultaneously. We define $S_{\Delta read_i^{D}}$ (similarly $S_{\Delta rst_i^{D}}$) as the set of packets in the traffic stream during $\Delta read_i^{D}$ (Δrst_i^{D}) where packets in this set update the counter array *before* the read and reset operation is executed.

The measured packets (the dotted line in Fig. 4) is $F((S_i - (S_{\Delta read_i} \cup S_{\Delta rst_i^{C1}} \cup S_{\Delta rst_i^{D}})) \cup (S_{\Delta read_{i+1}^{C1}} \cup S_{\Delta read_{i+1}^{D}}))$. Note that the effect of $\Delta read_i^{C2}$ is included in $S_{\Delta read_i}$. Next, we quantify the delays that cause the loss in accuracy.

[1] Based on the conversation with Barefoot, the Python API is not recommended for latency critical applications because Python API is a RPC wrapper for the C++ API.
[2] For the input packet stream S, we sample ten one-minute packet traces from inter-ISP packet trace captured on an OC-192 link [7].

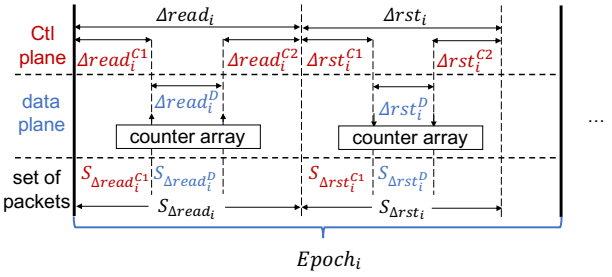


Figure 3: Decomposition of the read and reset delays into control plane and data plane delays at $Epoch_i$.

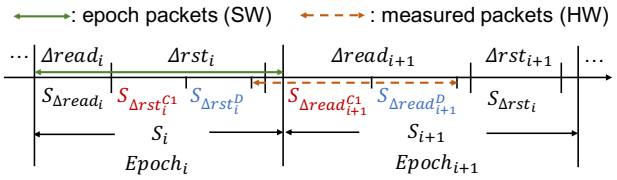


Figure 4: Different input packet sets between software and hardware create the discrepancy problem.

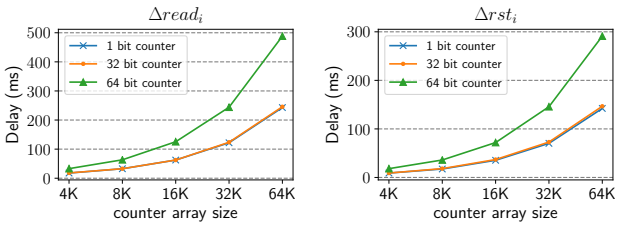


Figure 5: The read and reset delays (ms).

Delays		4K	16K	64K
$\Delta read_i$	$\Delta read_i^{C1}$	0.30	0.97	3.62
	$\Delta read_i^{D}$	0.01	0.07	0.31
	$\Delta read_i^{C2}$	22.21	66.70	244.49
	Total	**22.53**	**67.74**	**248.43**
Δrst_i	Δrst_i^{C1}	16.45	41.69	145.53
	Δrst_i^{D}	0.09	0.36	1.49
	Δrst_i^{C2}	0.02	0.02	0.03
	Total	**16.56**	**42.08**	**147.06**
$\Delta read_i + \Delta rst_i$		**39.10**	**109.81**	**395.48**

Table 1: Six delay measurement (ms).

3.2 Quantifying Sources of Error

To understand the magnitude of impact from each source of delay, we measure these next.

Methodology. At a high level, we measure the delays by sending packets at a controlled rate to the switch data plane and reading the counter values into the control plane. To this end, we use custom benchmarking programs in addition to the sketch implementations—data plane program using P4 language and the control plane using C++ API. Our measurements use efficient control plane read and reset operations. For the read operation, we utilize *table sync operation* which uses bulk DMA transfer from data plane counter arrays into control plane buffer so that the control plane can read counters more quickly. For the reset operation, we use the transaction API, which accelerates the individual write operations.

To measure $\Delta read_i^D$ we need to measure the time between when the control plane reads from the first counter *counter[0]* to the last counter *counter[N-1]* of a counter array. This is done by synthesizing a packet stream that contains two packets every $100\mu s$ and using them to increment (+1) *counter[0]* and *counter[N-1]* respectively. In this way, $\Delta read_i^D$ can be measured by (*counter[N-1]* $-$*counter[0]*) $\times 100\mu s$. The server uses `tcpreplay` to send this synthesized packet stream to the directly connected switch while the control plane executes the read operation.

We use the same setup to measure the duration of the data plane reset operation Δrst_i^D. However, since the reset operation resets counter values incrementally starting with the first one, the control plane executes the reset operation during `tcpreplay` and then executes the read operation after `tcpreplay` is finished. (*counter[0]* $-$*counter[N-1]*) $\times 100\mu s$ then represents Δrst_i^D. To measure $\Delta read_i^{C1}$ and $\Delta read_i^{C2}$, the control plane reads the first counter value before and after the read operation and we can then calculate those values using subtraction. We apply the same ideas for measuring Δrst_i^{C1} and Δrst_i^{C2}.

Result. Fig. 5 shows the $\Delta read_i$ and Δrst_i delays for different counter array sizes and counters (e.g., 1-bit, 32-bit). The read delay $\Delta read_i$ can be up to 488 ms and the reset delay Δrst_i can be up to 291 ms. Both delays increase linearly as the size of the counter array increases. For different counter sizes, a 64-bit counter takes 1.97× more delay than a 32-bit counter because the switch maintains a 64-bit counter as a pair of 32-bit counters. However, the delay difference between 32-bit and 1-bit counter is marginal (1.01×).[3]

Next, we look at six decomposed delays for 32-bit counters in Table 1. Surprisingly, $\Delta read_i^D$ and Δrst_i^D take less than 0.1%, 0.4% of the total read and reset delays. Meanwhile, we can see that $\Delta read_i^{C2}$ and Δrst_i^{C1} are the dominant factors as they take up more than 98% of the sum of the read and reset delays.

Key takeaways. Out of six delays, two *control plane* delays $\Delta read_i^{C2}$ and Δrst_i^{C1} are dominant factors. For example, $\Delta read_i^{C2}$ (Δrst_i^{C1}) of 16K array size takes 61% (38%) of the total sum of delays. Across all sizes of counter arrays, both bottleneck delays together account for 99% of the total delay.

4 Building Blocks and Solution Guidelines

In this section, we propose four solution building blocks to mask or reduce the delays identified in the previous section. These have varying trade-offs regarding the epoch size they can support, resource usage, general applicability across tasks. Table 2 summarizes

[3]We cannot measure the delays for 1-bit counters with the described methodology because 1-bit counter can not store an integer value. Instead, we used a timer in the control plane program to measure delays for the 1-bit counter in Fig. 5.

Building Blocks	$\Delta read_i^{C2}$	Δrst_i^{C1}	Epoch	Gen.	Res.
B1	hide	hide	smallest	✓	2x
B2	hide	hide	small	×	1x
B3	hide	×	med	✓	1x
B4	×	reduce	med	✓	1x

Table 2: Tradeoffs for solution building blocks in different metrics such as hiding/reducing two bottleneck delays, epoch size it can support, generality, resource usages.

these trade-offs. We also provide some general guidelines for combining building blocks as solutions appropriate for different use cases.

4.1 Building Blocks

B1: Use duplicate counters. A simple idea is to duplicate sketch instances in the data plane and alternately use them for odd/even epochs. At $Epoch_i$, counter array in sketch instance 1 can be updated in the data plane while the control plane reads and resets sketch instance 2. Then at $Epoch_{i+1}$, counter array in sketch instance 2 can be updated in the data plane while the control plane reads and resets instance 1.

Trade-off. This idea masks all delays and the key bottleneck delays. However, this idea requires 2× the data plane memory. Realizing it also requires some data plane code (P4) change.

B2: Offset counter errors in the control plane. Some sketches have a *linearity property* [33]. That is, counter arrays can be combined in a mathematical sense by addition and subtraction of each counter. In such cases, the control plane can avoid explicitly resetting the counters or duplicating the counters. Instead, it stores the counter arrays reported from the previous epoch (in the control plane) and obtains the counters for the current epoch by subtracting the previous counter arrays from counter arrays reported at the current epoch.[4]

Trade-off. This idea masks the key bottleneck delays of the reset and does not incur any additional data plane resources. It only requires small control plane code updates to subtract counter arrays. However, this idea is only applicable to sketches satisfying the linearity property. Fortunately, we've seen a range of linear sketches such as [15, 20, 26, 30, 35] for various measurements. We do see one caveat that some sketches for tracking heavy hitters in the data plane [18] need to access per-epoch counters to identify heavy flowkeys. Since the data plane only stores accumulated values, we cannot obtain per-epoch values directly in the data plane.

B3: Defer control plane read operation. We observe that during $\Delta read_i^{C2}$, most of the time is spent on reading counter arrays from an internal buffer in the control plane. That is, data is already transferred from the data plane using bulk DMA transfer as in Fig. 6. Thus, we can defer this operation of reading data from the buffer after the reset operation. We can implement this idea because

[4]The idea of not resetting the counters across epochs can bring up a concern of overflow. However, subtracting two counter array still works as long as there is at most 1 overflow per epoch. Empirically, the 32-bit counter is large enough to avoid two overflows.

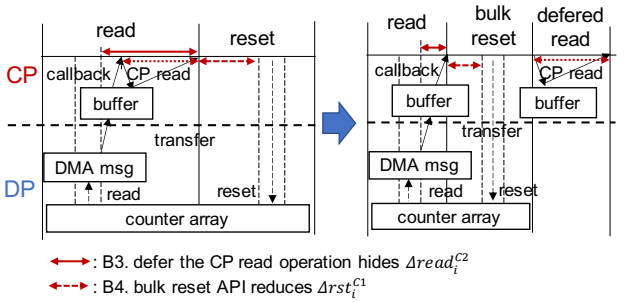


Figure 6: B3: Defer control plane read operation and B4: Use bulk reset API.

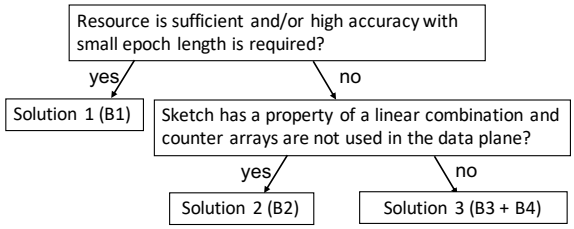


Figure 7: Decision tree for selecting solutions.

1) the reset operation does not reset the internal buffer and 2) the read operation can be divided into separate API calls: bulk DMA transfer and reading data from the internal buffer.

Trade-off. This idea does not require additional resources and it can be applied to sketches without linearity property. However, it only reduces the effect of $\Delta read_i^{C2}$.

B4: Use bulk reset API. This solution building block directly reduces Δrst_i^{C1} as in Fig. 6. We observed that the basic control plane support for reset updates counters one at a time. This is effectively a write operation and provides a more general capability to write an arbitrary value at a specific location. However, we note that there is also a `clear` API that suffices for our needs well since it resets all of the counter arrays to zero with much lower delay (18× faster).[5]

Trade-off. This idea only reduces the effect of Δrst_i^{C1}, thus it still can suffer accuracy degradation for a small epoch length.

4.2 Guidelines for Sketch Developers

Based on the above building blocks, we suggest a guideline for sketch developers on which solution is appropriate for different use cases summarized in the decision tree (Fig. 7):

- **Solution1 (B1)** would fit for small sketches and/or resources are sufficient. B1 provides the highest fidelity, especially for small epoch length.
- **Solution2 (B2)** uses low resource footprint. It is a simple solution for sketches satisfying linearity when counter arrays are not used in the data plane.
- **Solution3 (Combine B3 and B4)**. These two building blocks can be combined to tackle two bottleneck delays. The combined

[5]According to the conversation with Intel, Tofino2 supports an even faster bulk reset API.

solution requires some implementation effort but is general and is appropriate when resource overhead is critical.

5 Evaluation

Our evaluation demonstrates that (a) all solutions significantly reduce the error of the hardware implementation relative to the expected accuracy and (b) the implementation effort for the solutions is marginal in terms of additional lines of code.

5.1 Experimental Setup

Testbed. We use an Edgecore Wedge 100BF Tofino-based programmable switch and a server equipped with dual Intel Xeon Silver 4110 CPUs, 128GB RAM, and a 100Gbps Mellanox CX-4 NIC connected to the switch. We use Tofino SDE version 9.1.1 in our experiment. We send the trace to the switch from a directly connected server using `tcpreplay`.

Traces. We use sampled ten one-minute packet traces from CAIDA backbone traces capture at 1/21/16 Chicago [7].[6]

Sketches. We implement five sketches, MRB [19], HLL [21], count sketch (CS) [15], count-min sketch (CM) [18], and UnivMon (UM) [31] using P4 language. MRB uses 1-bit counters and the rest of the sketches use 32-bit counters. MRB and HLL use one counter array and CS, CM, UM use four counter arrays. MRB, HLL estimate cardinality, CS, CM estimate the average relative error of top-100 heavy hitter flow counts, and UM estimates entropy. Note that out of five sketches, CS, CM, UM satisfy the linearity property. We assume that we know all of the flowkeys for CS, CM, UM since identifying heavy flowkeys on the data plane is orthogonal to this work. We use P4 version of $P4_{16}$.

Metrics of difference. We consider three types of metrics:

- *Raw counters:* We consider both the total counter value difference = $\sum_i |expected[i] - actual[i]|$ and the relative counter difference = $\frac{\sum_i (expected[i] != actual[i])}{array\_size}$.
- *Sketch Errors:* Average Relative Error (ARE) is $\frac{1}{k}\sum_{i=1}^{k}\frac{|f_i - \hat{f}_i|}{f_i}$, where k is 100. f_i is true flow count, $\hat{f}_i$ is flow count estimate, and $f_i \geq f_{i+1}$ for any i. This metric is used for CS and CM. Relative Error (RE) is $\frac{|\text{True}-\text{Estimate}|}{\text{True}}$, where $True$ is true statistic value and $Estimate$ is estimated value. This metric is used for MRB, HLL, UM.
- *Delay:* We measure the sum of delays that corresponds to union and subtraction components in §3.1: $\Delta read_i + \Delta rst_i^{C1} + \Delta rst_i^{D} + \Delta read_{i+1}^{C1} + \Delta read_{i+1}^{D}$.

5.2 Error and Delay Reduction

Counter difference reduction. We first look at the counter difference reduction in Table 3. We use a fixed epoch length of 1 second. We can see that all solutions reduce almost all of the total counter value difference compared to unoptimized hardware implementation. Specifically, Sol 1 incurs no counter difference, and Sol 2, Sol 3 incur little counter differences. Note that the total counter value difference has a more direct effect on sketch accuracy than the relative counter difference.

[6]We also run experiments with other traces such as data center traces [12] and attack traces [8]. Results are similar thus, they are not shown.

	MRB	HLL	CS	CM	UM
A.size	64K	4K	64K	64K	128K
Unopt	1273/2%	91/1%	618K/73%	700K/76%	1030K/26%
Sol 1	0/0%	0/0%	0/0%	0/0%	0/0%
Sol 2	×	×	10K/7%	10K/7%	16K/5%
Sol 3	5/0%	3/0%	22K/12%	22K/13%	33K/8%

Table 3: Total counter value difference / relative counter difference for five sketches and three solutions using epoch=1s.

Array size		MRB 64K	HLL 4K	CS 64K	CM 64K	UM 128K
Expected Errors	**Ideal sketch**	1.6%	4.8%	0.7%	0.4%	2.8%
Actual Errors	**Unopt**	20.1%	6.2%	35.4%	34.8%	64.7%
	Sol 1	1.6%	4.8%	0.7%	0.4%	2.8%
	Sol 2	×	×	1.0%	0.7%	2.8%
	Sol 3	1.7%	4.8%	1.5%	1.1%	3.6%

Table 4: Expected errors vs. actual errors using epoch=1s.

	4K	**16K**	**64K**
Unopt	**39.39**	**110.84**	**399.39**
Sol 1	0 (100%)	0 (100%)	0 (100%)
Sol 2	0.32 (99.20%)	1.04 (99.06%)	3.94 (99.01%)
Sol 3	1.53 (96.11%)	4.66 (95.79%)	16.67 (95.83%)

Table 5: The sum of delays after applying solutions in ms (% of reduction compared to unoptimized).

Error reduction. Next, we look at the error reduction in Table 4. Compared to errors on unoptimized implementation, actual errors on all solutions are almost close to expected errors measured on software implementation.

Delay reduction. Table 5 shows that all solutions reduce delays significantly. Sol 1 does not incur any delays. Sol 2 can reduce delays by 99% across all counter array sizes. Sol 3 also reduces delays by 95%. Note that the delays after applying solutions are still linear to the counter array size.

Detailed measurement. We observe reductions for fixed array size and epoch length. We pick one sketch (CS) and look at the counter differences and error reductions for different array sizes and epoch lengths. Fig. 8 shows that as array size increases, the total counter value difference increases linearly, but it is constant over epoch lengths. Note that Sol 1 does not incur any counter differences across all array sizes and epoch lengths. Fig. 9 shows that the error gap between expected and un-optimized measurement is increasing as array size increases and epoch length decreases. All solutions effectively reduce this gap and they show similar errors as expected.

5.3 Implementation Effort

Table 6 shows additional lines of code for implementing solutions. Sol 1 requires P4 code change for duplicating instances and C++ control plane program change for reading instances alternatively. Code change for Sol 2 is in an offline processing program written

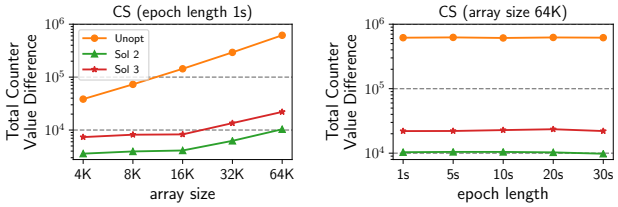


Figure 8: Total counter value difference for CS.

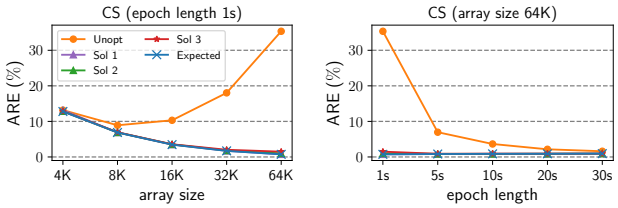


Figure 9: Average relative error for CS.

Additional Lines of code	Sol 1 B1	Sol 2 B2	Sol 3 B3	Sol 3 B4
Data Plane P4 Code	29	0	0	0
Control Plane Program (C++ API)	63	0	0	19
Offline Processing	0	9	0	0

Table 6: Additional lines of code for implementing solutions.

in Python for subtracting counter arrays. B3 in Sol 3 does not incur any additional lines of code since it just swaps the order of the control plane read and reset operation. B4 in Sol 3 requires additional control plane program code for bulk reset API.

6 Related work

Sketch-based telemetry. Sketches have emerged as a promising telemetry solution for flow-level measurements, including heavy hitters [15, 18, 31, 37, 39], entropy estimation [31, 34, 36], change detection [30, 40], and distinct flows [21, 31]. While recent efforts [14, 41] propose to maintain more light-weight sketches per device, they still suffer from the incorrect counter retrieval issue in programmable switches and can benefit from our solutions.

Other work in network telemetry. There are complementary telemetry capabilities that focus on packet-level and path-level monitoring (e.g., INT [29] and PINT [11]), higher-order telemetry (e.g., performance statistics [16, 22, 32], application level monitoring [38]), diagnosis [25, 28], as well as network-wide adaptive telemetry [23, 24]. A not hard extension is to explore if these telemetry tasks can suffer from a similar incorrect state retrieval and reset problem.

Other programmable platforms. In addition to the switches discussed in this paper, SmartNICs such as multicore SoC NICs [4, 5] and FPGA NICs [9] are platforms for telemetry. Recent work [27] in measuring the performances of various SmartNICs demonstrated a similar bottleneck between the data plane and the control plane. A future direction is to explore the telemetry retrieval inaccuracy problem in SmartNICs.

7 Conclusions

We consider a practical problem of deploying network telemetry tasks on programmable switches. We identify and quantify the causes of an accuracy degradation in sketches. Our solutions informed by our analysis can eliminate almost all the inaccuracy for five sketches. We believe our insights are more broadly applicable to other network telemetry tasks with similar control-data plane interactions.

Acknowledgement

We would like to thank the anonymous SOSR reviewers, Jeongkeun Lee, and Georgios Nikolaidis for their constructive feedback. This work was supported in part by the CONIX Research Center, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA, and by NSF awards 1565343 and 1700521.

References

[1] Barefoot Tofino. https://barefootnetworks.com/products/brief-tofino/.
[2] Barefoot P4 Studio. https://www.barefootnetworks.com/products/brief-p4-studio/.
[3] Broadcom Trident 4. https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm56880-series.
[4] Mellanox DPU. https://www.nvidia.com/en-us/networking/products/data-processing-unit/.
[5] Netronome Agilio SmartNICs. https://www.netronome.com/products/nfe/.
[6] tcpreplay. https://tcpreplay.appneta.com/wiki/tcpreplay-man.html.
[7] The CAIDA UCSD Anonymized Internet Traces. https://www.caida.org/data/passive/passive_dataset.xml.
[8] The U.S. National CyberWatch Mid-Atlantic Collegiate Cyber Defense Competition (MACCDC). https://www.netresec.com/?page=MACCDC.
[9] Xilinx FPGA. https://www.xilinx.com/products/silicon-devices/fpga.html.
[10] Ben Basat, R., Einziger, G., Friedman, R., Luizelli, M. C., and Waisbard, E. Constant time updates in hierarchical heavy hitters. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (2017).
[11] Ben Basat, R., Ramanathan, S., Li, Y., Antichi, G., Yu, M., and Mitzenmacher, M. Pint: probabilistic in-band network telemetry. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication* (2020), pp. 662–680.
[12] Benson, T., Akella, A., and Maltz, D. A. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement* (2010), pp. 267–280.
[13] Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., and Walker, D. P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.* (2014).
[14] Bruschi, V., Basat, R. B., Liu, Z., Antichi, G., Bianchi, G., and Mitzenmacher, M. Discovering the heavy hitters with disaggregated sketches. In *Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies* (2020), pp. 536–537.
[15] Charikar, M., Chen, K., and Farach-Colton, M. Finding frequent items in data streams. In *International Colloquium on Automata, Languages, and Programming* (2002), Springer, pp. 693–703.
[16] Chen, X., Kim, H., Aman, J. M., Chang, W., Lee, M., and Rexford, J. Measuring tcp round-trip time in the data plane. In *Proc. of SIGCOMM SPIN Workshop* (2020).
[17] Chen, X., Landau-Feibish, S., Braverman, M., and Rexford, J. Beaucoup: Answering many network traffic queries, one memory update at a time. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication* (2020), pp. 226–239.
[18] Cormode, G., and Muthukrishnan, S. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms 55*, 1 (2005), 58–75.
[19] Estan, C., Varghese, G., and Fisk, M. Bitmap algorithms for counting active flows on high speed links. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement* (2003), pp. 153–166.
[20] Fan, L., Cao, P., Almeida, J., and Broder, A. Z. Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM transactions on networking 8*, 3 (2000), 281–293.
[21] Flajolet, P., ric Fusy, Gandouet, O., and et al. Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm. In *AOFA* (2007).

[22] Ghasemi, M., Benson, T., and Rexford, J. Dapper: Data plane performance diagnosis of tcp. In *Proceedings of the Symposium on SDN Research* (2017), pp. 61–74.

[23] Gupta, A., Harrison, R., Canini, M., Feamster, N., Rexford, J., and Willinger, W. Sonata: Query-driven streaming network telemetry. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication* (2018), pp. 357–371.

[24] Harrison, R., Cai, Q., Gupta, A., and Rexford, J. Network-wide heavy hitter detection with commodity switches. In *Proceedings of the Symposium on SDN Research* (2018), pp. 1–7.

[25] Holterbach, T., Molero, E. C., Apostolaki, M., Dainotti, A., Vissicchio, S., and Vanbever, L. Blink: Fast connectivity recovery entirely in the data plane. In *Proc. of NSDI* (2019).

[26] Huang, Q., Lee, P. P., and Bao, Y. Sketchlearn: Relieving user burdens in approximate measurement with automated statistical inference. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication* (2018), pp. 576–590.

[27] Katsikas, G. P., Barbette, T., Chiesa, M., Kostic, D., and Maguire Jr, G. Q. What you need to know about (smart) network interface cards. In *PAM* (2021).

[28] Khandelwal, A., Agarwal, R., and Stoica, I. Confluo: Distributed monitoring and diagnosis stack for high-speed networks. In *Proc. of USENIX NSDI* (2019).

[29] Kim, C., Sivaraman, A., Katta, N., Bas, A., Dixit, A., and Wobker, L. J. In-band network telemetry via programmable dataplanes. In *ACM SIGCOMM Demo Session* (2015).

[30] Krishnamurthy, B., Sen, S., Zhang, Y., and Chen, Y. Sketch-based change detection: methods, evaluation, and applications. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement* (2003), pp. 234–247.

[31] Liu, Z., Manousis, A., Vorsanger, G., Sekar, V., and Braverman, V. One sketch to rule them all: Rethinking network flow monitoring with univmon. In *Proceedings of the 2016 ACM SIGCOMM Conference* (2016), pp. 101–114.

[32] Liu, Z., Zhou, S., Rottenstreich, O., Braverman, V., and Rexford, J. Memory-efficient performance monitoring on programmable switches with lean algorithms. In *Proc. of APoCS* (2020), SIAM.

[33] Muthukrishnan, S. *Data streams: Algorithms and applications*. Now Publishers Inc, 2005.

[34] Nychis, G., Sekar, V., Andersen, D. G., Kim, H., and Zhang, H. An empirical evaluation of entropy-based traffic anomaly detection. In *ACM IMC, 2008*.

[35] Nychis, G., Sekar, V., Andersen, D. G., Kim, H., and Zhang, H. An empirical evaluation of entropy-based traffic anomaly detection. In *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement* (2008), pp. 151–156.

[36] Nychis, G., Sekar, V., Andersen, D. G., Kim, H., and Zhang, H. An empirical evaluation of entropy-based traffic anomaly detection. In *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement* (2008), pp. 151–156.

[37] Sivaraman, V., Narayana, S., Rottenstreich, O., Muthukrishnan, S., and Rexford, J. Heavy-hitter detection entirely in the data plane. In *Proceedings of the Symposium on SDN Research* (2017), pp. 164–176.

[38] Wang, L., Kim, H., Mittal, P., and Rexford, J. Programmable in-network obfuscation of dns traffic (work-in-progress).

[39] Yang, T., Jiang, J., Liu, P., Huang, Q., Gong, J., Zhou, Y., Miao, R., Li, X., and Uhlig, S. Elastic sketch: Adaptive and fast network-wide measurements. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication* (2018), pp. 561–575.

[40] Yu, M., Jose, L., and Miao, R. Software defined traffic measurement with opensketch. In *Proc. of USENIX NSDI* (2013).

[41] Zhao, Y., Yang, K., Liu, Z., Yang, T., Chen, L., Liu, S., Zheng, N., Wang, R., Wu, H., Wang, Y., et al. Lightguardian: A full-visibility, lightweight, in-band telemetry system using sketchlets. In *18th USENIX Symposium on Networked Systems Design and Implementation* (2021).