IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 4, 2019, accepted October 8, 2019, date of publication October 11, 2019, date of current version October 22, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2946704

Interleaved Sketch: Toward Consistent Network Telemetry for Commodity Programmable Switches

ZIJUN HANG, MEI WEN, YANG SHI, AND CHUNYUAN ZHANG
School of Computer Science, National University of Defense Technology, Changsha 410073, China

Corresponding author: Zijun Hang (hangzijun17@nudt.edu.cn)

This work was supported in part by the National Key Research and Development Program under Grant 2016YFB1000400, and in part by the National Nature Science Foundation of China through NSFC under Grant 61502509 and Grant 61402504.

ABSTRACT Network telemetry is vital to various network applications, including network anomaly detection, capacity planning, and congestion alleviation. State-of-the-art network telemetry systems are claimed to be scalable, flexible, all-purpose, and accurate. They adopt interval approaches that track network traffic in each interval and collect statistics for analysis at a specific epoch. However, interval methods are impaired by collecting inconsistency and clearing inconsistency, which pollute statistics. Moreover, The state-of-the-art centralized controllers have long latency, which aggravates the discrepancy. Accordingly, we propose the interleaved sketch, a consistent and decentralized network telemetry system across all switches. Each switch has two asymmetric sketches that work in an interleaved fashion, and is self-supervised to improve consistency. The distributed control plane extracts the flow characteristics and provides network-wide telemetry with low latency. We build a P4 prototype of our proposed interleaved sketch and test it on a Barefoot Tofino switch. Experimental results demonstrate that our interleaved sketch achieves ideal accuracy at line speed, with 6% resource overhead.

INDEX TERMS Network telemetry, P4, programmable switches, software defined network.

I. INTRODUCTION

Network telemetry provides a network-wide perspective by monitoring massive network traffic. Network telemetry is the cornerstone of many network applications, including congestion control, anomaly detection [1]–[3], and Heavy Hitter detection [4], [5], as it is highly beneficial to these applications. To ensure accuracy, network telemetry systems should present a network-wide and consistent view that covers as many switches as possible and provides accurate flow-level statistics.

Conventional wisdom mainly focuses on single switches and single task. However, this focus is not general enough for a wide range of telemetry tasks, as well as lacking in completeness in terms of the network-wide telemetry. Most importantly, they focuses primarily on CPUs [6], which are unable to handle data center traffic [7].

The associate editor coordinating the review of this manuscript and approving it for publication was Wangli He.

Software-Defined Network (SDN) [8] represented a revolution in traditional network conception and has opened up a new era in network telemetry. SDN decouples the data plane from the control plane of switches and makes the data plane programmable, which makes processing while forwarding each packet in line-rate a reality. The ongoing SDN revolution has facilitated the innovation of telemetry algorithms, including FlowRadar [9], UnivMon [10], SketchLearn [11], and Elastic Sketch [12]. These telemetry algorithms adopt interval approaches that track network traffic in each interval and collect statistics for analysis at the end of each interval.

However, these approaches may lack consistency in three key respects. First, at the end of each interval, the controller collects statistics from the data plane, which takes 0.1-3 seconds according to our experimental data. It is impossible to halt the data plane while the controller is collecting statistics, as traffic alters the statistics of the data plane as it continues to arrive. The statistics that are read later will be larger, which contributes to inconsistency in the statistic. Algorithms such as SketchLearn and Elastic Sketch are sensitive to

such inconsistency. Second, after the collecting procedure, the controller needs 0.1-1 second to clear the statistics according to our experiments. The statistics are cleared at different epochs; statistics that are cleared earlier will be more significant. Third, many telemetry systems employ a centralized controller to provide a network-wide view. However, the centralized controller has undetermined latency, which contributes to the inconsistency between controller and switch.

Accordingly, to overcome these defects, we present the interleaved sketch, a consistent and decentralized network telemetry system across all switches. The interleaved sketch has two sketch pipelines for the data plane and control plane to facilitate access in an interleaved way — in brief, the data plane and control plane interchange access to two pipelines from interval to interval. Moreover, we decouple the control plane and management plane in order to implement a decentralized telemetry system in which each switch is self-supervised. We move the control plane down to each switch so that each control plane automatically controls the interplay of the two pipelines inside a switch. The control plane also extracts flow characteristics from the data plane statistics inside each switch. Our proposed protocol is designed to transmit the flow characteristics to provide network-wide telemetry.

Realizing the proposed interleaved sketch in a commodity programmable switch is challenging. First, the concept of an interleaved sketch calls for coordination between the data plane and the control plane. The control plane must finish all transactions before the data plane changes the working pipeline to prevent the statistics from being inconsistent. Second, the decentralized controller increases the complexity of both programming and configuring the switches. Third, programmable switches have tight restrictions on resources and transactions; naively duplicating the pipeline is therefore not feasible. Fourth, the volume of statistics can be 200KB-1500KB [9]–[12]. Reading them is time-consuming, not to mention processing them.

The unique characteristics of the proposed interleaved sketch are as follows

- The interleaved sketch does not pollute the statistics. Instead, it explores the interleaved working of two pipelines to support consistency among the statistics. The two pipelines adopt asymmetric sketches; this method consumes only 6% more resources than a single pipeline.
- The interleaved sketch is flexible and can be scaled to provide network-wide telemetry on commodity programmable switches. Each switch is self-supervised, enabling pipeline access to be interchanged without manual intervention.
- The interleaved sketch stresses low latency by distributed analysis and by only reporting the final results rather than all raw statistics.

The remainder of this paper is organized as follows. Section II presents the background and motivation. Section III describes the design and implementation of the interleaved sketch. Section IV evaluates the performance of the interleaved sketch. Section V provides an outline of related works. Finally, we conclude this paper in Section VI.

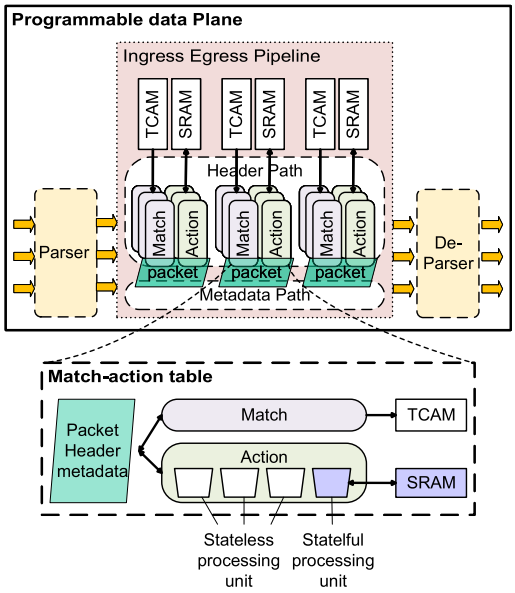


FIGURE 1. PDP outline.

II. BACKGROUND AND MOTIVATION

A. PROGRAMMABLE SWITCH

Programmable switches stimulate the innovation of telemetry systems. The emerging Programmable Data Plane (PDP) [13] and P4 (Programming Protocol-independent Packet Processors) [14] technologies are critical to programmable switches. Figure 1 presents the outline of PDP. Instead of targeting a specific protocol and function, PDP is reconfigurable, thanks to the Reconfigurable Match-action Tables (RMT) [15] architecture. P4 is a domain-specific language (DSL) [16] that can describe the packet processing behavior of PDP. Programmers can reconfigure the PDP behavior by deploying a new program that governs the operation of the RMT pipeline like software. Commodity programmable switches [13], [17], [18] are flexible in terms of their PDP and comparable with fixed switches [19], [20] in performance.

To be more specific, the RMT pipeline is programmable in three key respects:

1) The parser can be programmed to identify custom packet headers.
2) The match-action tables can be programmed to perform custom matching and actions.
3) The on-chip SRAMs can be programmed so that state information persists across packets.

In this paper, we leverage the Barefoot Tofino [13] programmable edge switch to implement our interleaved sketch. This is not easy, as Tofino imposes rigid limitations:

1) Limited Stages. Tofino only has 12 stages in the RMT pipeline. Thus, all transactions must be accomplished in 12 stages.
2) Simple Operations. Each stage can only perform simple operations provided by P4 primitives. Only primitive arithmetic (e.g. addition and subtraction) is allowed in each stage.
3) Limited Concurrent Memory Access. Each stage can access a few memory locations of different register arrays, but only one location of a register array. Two or more stages cannot access the same memory location.

B. INTERVAL METHODS

This paper targets flow-level telemetry systems that identify flow by means of flow key and track flow statistics. These systems are interval methods under strict turnstile model [21]. Interval methods monitor traffic and collect traffic statistics over time intervals called epochs.

Interval methods are a popular type of network algorithms due to their fast response, space efficiency, and network-wide measurement capability. These methods are sensitive to micro-bursts and respond quickly, with intervals in seconds or milliseconds.

Interval methods use sketches to compress data representation, which saves the limited space available on switch chips. Sketches process every packet in a stream while analyzing the underlying data, using the emerging programmable switch. Furthermore, sketches consume limited memory space to measure network flows with guaranteed estimation accuracy.

The strict turnstile model is an abstraction of sketch structure. In the model, the packets $p_1, p_2, \cdots$ arrive sequentially in a stream, and A is a register array that tracks flows. The i^{th} packet $p_i = (j, I_i)$ results in $A_i[j] = A_{i-1}[j] + I_i$, where A_i is the register array after the i^{th} packet of the stream has been seen.

The interval methods also provide a network-wide measurement. At the end of each epoch, the centralized controller collects statistics from all switches and analyzes the network-wide situation. The centralized controller integrates the functions of the control plane and the management plane.

C. INCONSISTENCY PROBLEM FOR INTERVAL METHODS

SketchLearn [11] is an interval method characterized by its scalability, flexibility, generality, and accuracy. Figure 2 presents the basic concept underpinning the SketchLearn algorithm. The distributed PDPs track the flow statistics in multilevel sketches. The centralized controller collects statistics from all PDPs and analyzes network-wide large flows using the statistics. Figure 3(a) provides an overview of the data plane transactions of SketchLearn. SketchLearn tracks the flow key distribution for each incoming packet in a multilevel sketch. Assuming that the flow key length is k bits, SketchLearn uses a $k+1$-level sketch. Each level is a $r \times c$ register array where the position (i, j), $(0 \le i < r, 0 \le j < c)$ is indexed by $hash_i(key) = j$. When the packet with flow

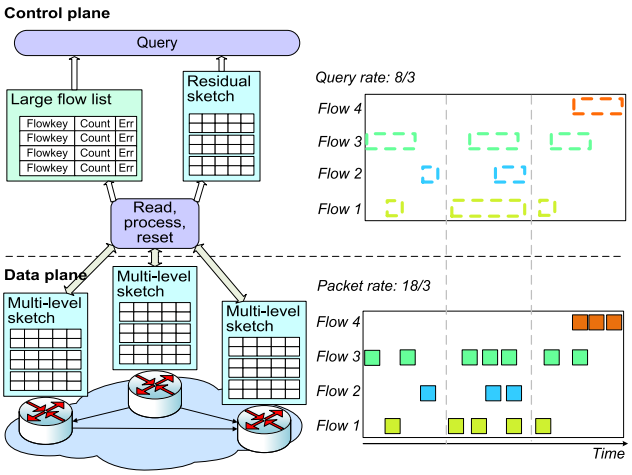
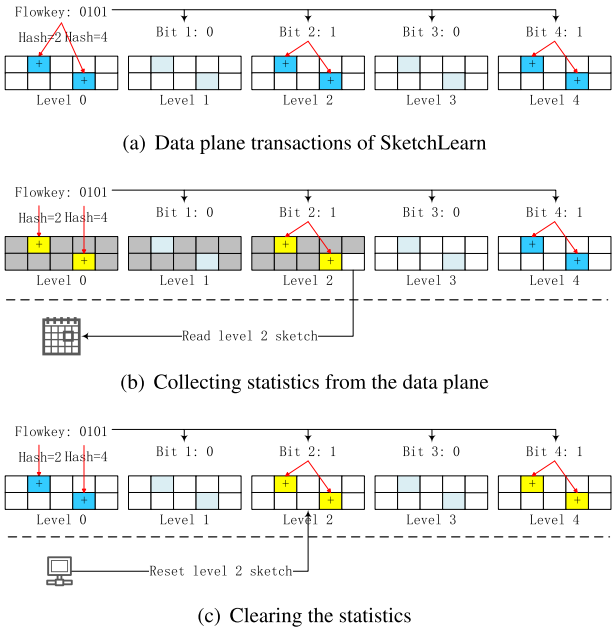


FIGURE 2. SketchLearn with centralized controller.



(a) Data plane transactions of SketchLearn

(b) Collecting statistics from the data plane

(c) Clearing the statistics

FIGURE 3. Inconsistency illustration of SketchLearn.

key 0101 comes, SketchLearn calculates two hash indexes, $hash_0(0101) = 1$ and $hash_1(0101) = 3$. It then updates position (0,1) and (1,3) in levels 0, 2, and 4.

SketchLearn is not directly applicable to the industrial circumstance for inconsistency. Figure 3(b) shows the **collecting inconsistency**. At the end of each interval, the controller collects statistics from the multi-level sketch. The controller reads every slot in all levels serially. In the figure, the controller has read the gray slots and is reading position (1, 4) of level 2. At this time, a packet with flow key 0101 arrives, updating the level 0, 2, and 4 sketches. For levels 0 and 2, the values of which have already been read, the newly updated values are lost. Figure 3(c) shows the **clearing inconsistency**. Unlike in the collecting process, the controller uses a reset instruction that clears one level at a time. When a

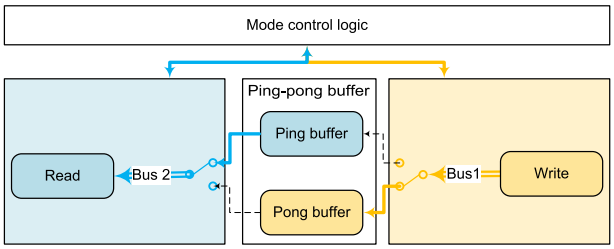


FIGURE 4. Ping-pong buffer outline.

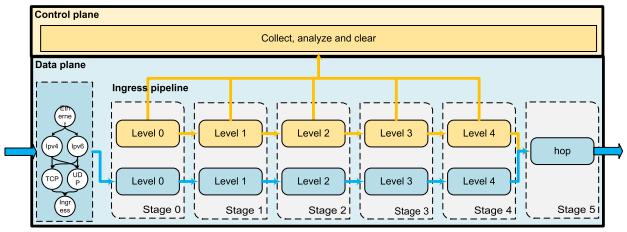


FIGURE 5. Interleaved sketch outline.

new flow comes, the controller clears the level 2 sketch; thus, the updates in levels 2 and 4 are lost.

The collecting process lasts for 0.1-3 seconds depending on the size of the sketch array (r, c and k). E.g., with $r = 1, c = 256, k = 32$, it takes an average of 2 seconds to read all sketches. The controller reads level 0 at time t while reading level 32 at time $t+2$, deviating the statistics from time t. This deviation is analogous to a clearing process where it takes 0.1-1 second to clear all sketches. Assuming the link speed is 40Gbps and average packet size is 1KB, 5M packets are lost in one second. A flood may occur in this period; if this occurs, it will not be detected. Worse yet, level 32 tracks 40M more packets than level 0. SketchLearn cannot extract flows from inconsistent data. This deviation is thus catastrophic for SketchLearn, which relies on a Gaussian distribution of all flow keys.

III. DESIGN OF INTERLEAVED SKETCH

A. SOLUTIONS FOR INCONSISTENCY

Since it is impossible to halt the data plane until the controller collects all statistics, is there a better way to enable interval methods on commodity switches? We implement two adaptations to the interval methods: interleaved sketch and decentralized controller.

1) INTERLEAVED SKETCH

Inspired by the ping-pong buffer mechanism [22], which supports support parallel reads and writes on storage, we propose the interleaved sketch method, to support interval methods. The ping-pong buffer supports reading and writing simultaneously. Figure 4 shows the ping-pong buffer outline. The ping-pong buffer utilizes dual-port storage: this involves a single storage array and two independent ports, where each port has separate data, address, and control lines. Write logic is coupled to one of the independent ports for receiving data from one of the data buses and writing it into the first portion of the storage array. Read logic is coupled to the other independent ports to facilitate simultaneously reading data from a second portion of the storage array and supplying it to the other data bus. Mode control logic enables the writing and reading functions of the first and second portions of the storage array to be interchanged back and forth from time to time, allowing data may be read from one portion while being written into the other portion and vice versa.

The proposed interleaved sketch shares some similarities with the ping-pong buffer mechanism, as illustrated in Figure 5:

1) They both have single logical storage for simultaneous reading and writing.
2) They both have a write logic. The write logic in the interleaved sketch is the data plane.
3) They both have a read logic. The read logic in the interleaved sketch is the control plane.

Meanwhile, they also have some differences:

1) The physical storage is different. Each portion of the ping-pong buffer storage has several blocks. Physically, the two portions make up a whole storage bank. The interleaved sketch disperses the storage into two pipelines, each of which further disperses the storage into chained stages.
2) The ping-pong buffer has global storage and addresses it using address lines. The interleaved sketch has local storage in each stage and addresses it using a Stateful Arithmetic and Logical Unit (SALU).
3) The interleaved sketch has no specific mode of control logic. Instead, the data plane and control plane together play the role of control logic.

The data plane and control plane jointly enable the writing and reading functions of the first and second pipeline of the interleaved sketch to be interchanged back and forth from interval to interval. Thus, the control plane reads statistics from one pipeline while the data plane updates the other pipeline, and vice versa.

2) DECENTRALIZED CONTROLLER

Prior works have integrated the control plane and the management plane into a centralized controller. However, we argue that a decentralized controller is necessary. As Figure 2 shows, when the centralized controller collects statistics from all PDPs, the PDPs transmit a large number of raw statistics to the centralized controller, which aggravates the network congestion. This congestion augments the latency between the PDPs and the controller, which contributes to late decisions.

Conventional wisdom advocates for a focus on time synchronization issue to strengthen the consistency. We demonstrate that lower communication overhead is another crucial property for improving consistency, as lower overhead reduces both congestion and latency.

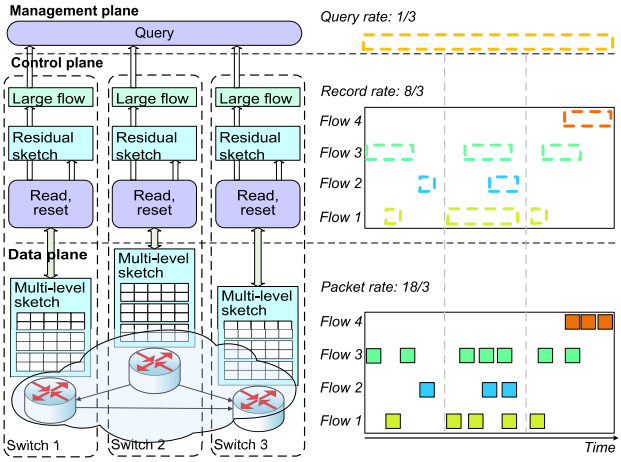


FIGURE 6. Decentralized controller.

In our approach, which involves a decentralized controller, we decouple the control plane and the management plane and implement a decentralized control plane across all switches. We identify three entities: the data plane (PDP), the control plane, and the management plane. As illustrated in Figure 6, we implement the collecting, analyzing, and clearing functions inside the control plane of each switch. Each switch is self-supervising through the automatic interplay of its data plane and control plane. The control plane of each switch reports to the management plane and thereby provides a network-wide view.

By reporting only large flows to the management plane rather than all statistics, we reduce the overhead by several orders of magnitude. Assuming that the packet rate is 18 out of 3 epochs, the query rate for the centralized control plane is 8/3, as shown in Figure 2. By dispersing the large flows into decentralized control planes, we reduce the query rate to 1/3; in other words, one packet fetches all large flow records of the three epochs, as shown in Figure 6.

One concern at this point is that the decentralized controller will increase the complexity of the switch configuration. Works such as Sluice [23], SNAP [24], and P4HLPc [25] explore the configuration of multiple switches in a network. We can use these works to resolve the complexity of configuring switches.

B. CHARACTERIZING STORAGE

The P4 [26] specification defines three types of storage: register, meter, and counter. They ensure that states persist across packets in PDP. In this paper, we focus on the register, which is widely adopted in network algorithms. We identify three usages of registers:

- PDP register, which PDP writes to track packet measurements.
- control plane registers, which the control plane reads to collect statistic.
- flag register, which indicates PDP and control plane status, e.g., the flag register to interleave sketches

Unlike the flag register, the PDP register and control plane register are exchangeable in different epochs, owing to the interleaved nature of PDP and the control plane.

We need to migrate register values from PDP to the control plane for analysis, as the data plane can only carry out simple operations and is unable to read too many register entries. Either PDP or the control plane can migrates register values.

The PDP can migrate register values through Inband Network Telemetry (INT), which is a framework for collecting and reporting network states by means of the data plane, without the intervention of the control plane. Packets contain header fields, which function as telemetry instructions that tell the device what state to collect and write into the packet as it transits the network. Compared to traditional approaches such as Netflow [6], INT is more flexible, as it does not restrict the PDP measurement method. Moreover, INT is independent of the underlying devices; furthermore, all programmable switches from different vendors can support INT, while Netflow targets Cisco devices. Most importantly, Netflow relies on CPU sampling, which is inaccurate and inefficient. However, we also need to take the transmission time into account when migrating register values. INT has undetermined latency, as it relies on migration path traffic. Moreover, any one packet can read only a few locations (i.e., no more than 48 in Tofino). This leads to aggravated network congestion when a mass of register arrays are being read, e.g., SketchLearn uses 30000 registers. The PDP migration method is applicable for applications that are insensitive to latency and read only a few values.

The control plane can read register values directly, and does so through a runtime API generated by a P4 compiler. The migration takes place inside each switch, independent of network state and other entities. The migration inside the switch is fast and reliable.

The migration time is the overall time required to migrate all registers in one sketch before it can work for the next epoch. Intuitively, the total migration time will increase with the number of register entries, regardless of which migration approach we apply. We compare these two methods in Section IV.

C. DATA PLANE TRANSACTIONS

As a general rule, telemetry systems should never exhaust any specific resources that could otherwise be used by conventional packet processing. Naively duplicating the pipeline is impossible for resource-consuming algorithms given the finite resources available in the PDP. For example, Tofino PDP binds a SALU to a register array. The PDP accesses the register array through the SALU. Sketchlearn costs 32 out of all 48 SALUs in PDP to identify 32-bit flow keys. The remaining SALUs is not sufficient to create another SketchLearn pipeline. Moreover, it is unnecessary to duplicate the pipeline. The control plane reads register arrays in 0.1-3 seconds. A lightweight sketch in PDP interleaving with the regular sketch is accurate enough within this period.

TABLE 1. Interval algorithm overview.

Algorithm	Application
SpaceSaving HashPipe RAP PRECISION	Heavy hitter detection
FlowRadar	Per-flow frequency
UnivMon	Heavy hitter detection Heavy changers Cardinality estimation Frequency distribution and entropy
SketchLearn	
Elestic sketch	

Algorithm 1 SketchLearn Algorithm

Input: Packet p with $iKey$
for $i \leftarrow 0$ **to** $r-1$ **do**
  $j \leftarrow hash_i(iKey)$;
  $reg[i][j][0] \leftarrow reg[i][j][0]+1$; ▷level 0
  for $k \leftarrow 1$ **to** k **do**
    if $(iKey \gg k)$ & *0x01 != 0* **then**
      $reg[i][j][k] \leftarrow reg[i][j][k]+1$; ▷level k

We adopt an asymmetric sketch architecture in PDP, which consists of a regular sketch and a lightweight sketch. The regular sketch consumes more resources than the lightweight sketch and is also more accurate. The lightweight part is responsible for temporarily measuring PDP flows when the control plane accesses the regular sketch.

There are two categories of asymmetric sketch. One adopts the same algorithm for the regular sketch and the lightweight sketch, and the control plane carries out the same analyzing routine for the two sketches; the only difference is in the size of the two sketches. The other adopts different algorithms for the two sketches, such that the control plane and data plane both change their operations among different epochs.

Table 1 summarizes some interval algorithms and their applications. This paper adopts some principles from these existing sketch algorithms. Single purpose algorithms, e.g., SpaceSaving (SS) [4], HashPipe (HP) [5], RAP [27], and PRECISION (PC) [28] Heavy Hitter (HH) detection algorithms, measure heavy hitters with key-value data structures. General-purpose algorithms, e.g., FlowRadar (FR) [9], UnivMon (UM) [10], SketchLearn (SL) [11], and Elastic Sketch (ES) [12], measures flow level statistics with complicated data structure and can benefit multiple applications. A heavy hitter denotes a flow size that exceeds a threshold in an epoch. Heavy change is a flow whose size change across two consecutive epochs exceeds a threshold. Cardinality refers to the number of distinct flows in an epoch. Flow size distribution is the fractions of flows for different ranges of byte counts in an epoch. Entropy denotes the entropy of flow size distribution in an epoch.

We choose SketchLearn (Algorithm 1) for the regular sketch and PRECISION (Algorithm 2) for the lightweight

Algorithm 2 PRECISION Heavy Hitter Algorithm

Input: Packet p with $iKey$
initialization: $carry\_min \leftarrow 0x7fffffff$;
if *packet is cloned* **then**
  $i \leftarrow min\_stage$; ▷for cloned packets, update key and value
  $l_i \leftarrow hash_i(iKey)$;
  $Key_i[l_i] \leftarrow iKey$;
  $Val_i[l_i] \leftarrow new\_val$;
  Drop;
for $i \leftarrow 1$ **to** d **do**
  $l_i \leftarrow hash_i(iKey)$; ▷for normal packets, find the minimum bucket
  if $Key_i[l_i] == iKey$ **then**
    $matched \leftarrow true$;
    $Val_i[l_i] \leftarrow Val_i[l_i]+iVal$;
  else if $Val_i[l_i] < carry\_min$ **then**
    $carry\_min \leftarrow oval_i$;
    $min\_stage \leftarrow i$;
if $\neg matched$ **then**
  $new\_val \leftarrow 2^{\lceil log_2(carry\_min)\rceil}$; ▷decide replace or not
  $R \leftarrow random(0, new\_val)$;
  if $R = 0$ **then**
    clone and recirculate packet with new_val, min_stage;

Algorithm 3 PDP Transactions

Input: Packet p with $iKey$
initialization: clear flag register and all registers of SketchLearn, PRECISION;
while *packet p arrives*
  $tmp \leftarrow read\_flag(0)$;
  if $tmp == 0$ **then**
    SketchLearn(p) ; ▷flag is 0:SketchLearn
  else if $tmp == 1$ **then**
    PRECISION(p) ; ▷flag is 1:PRECISION

sketch in order to illustrate how the interleaved sketch works. PRECISION has more complicated PDP transactions. PRECISION consists of two paths, namely the regular path and the recirculation path. For each incoming packet, the regular path updates the keys and values in d stages according to the hash indexes. If the packet has no matching key in any stages, the packet is recirculated at a probability of $\frac{1}{2^{\lceil log_2(carry\_min)\rceil}}$. For its part, the recirculation path updates the minimum key and value according to the metadata in the recirculated packet and drops the cloned packet.

Algorithm 3 presents the overall PDP transactions of the asymmetric sketch. We use a one-bit register flag to indicate pipeline access. PDP works on SketchLearn when the flag

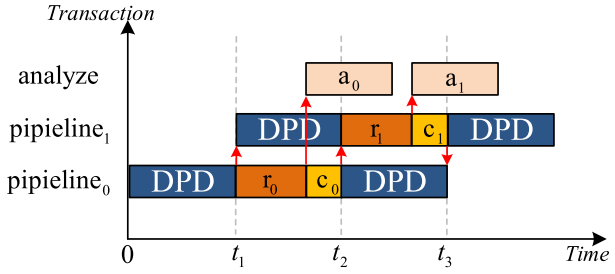


FIGURE 7. PDP and Control plane coordination.

is 0, while the converse is true for PRECISION. To avoid loss of generality, we evaluate different combinations of these algorithms in Section IV.

D. CONTROL PLANE TRANSACTIONS

The P4 compiler generates P4Runtime [29] APIs, i.e., interfaces for controlling the data plane elements, to enable the control plane functions. The control plane executes a PTF (Packet Test Framework) [30] script to invoke APIs that are exposed over Thrift-RPC [31], including port management, table entry modification, and register operation. This paper targets the register operation APIs.

Algorithm 4 provides an outline of the control plane transactions. Lines 35-43 show the routine of the control plane. First, it reads the flag register value (line 36). If the flag is 0, which indicates that the PDP is running SketchLearn, the control plane should process the PRECISION statistics (lines 37-38); otherwise, the control plane should process the SketchLearn statistics (lines 39-40). As the process routine may finish before the user-defined epoch, so the control plane waits until timeout (line 41), after which the controller reverses the flag register (line 42).

We identify three routines of the control plane: register read (r_0 and r_1), register clearing (c_0 and c_1), and sketch analysis (a_0 and a_1). The subscripts 0 and 1 indicate SketchLearn and PRECISION transactions, respectively. In r_0 and r_1, the control plane collects sketch statistics from PDP through the *read_reg* API. For both algorithms, the controller reads every slot of one register array serially, then moves to the next array, due to a lack of multi-thread support in thrift. After reading all values, the control plane clears the register arrays using the *reset_reg* API in c_0 and c_1. The API clears one entire register array at a time. In a_0 and a_1, the control plane analyzes the sketch statistics. SketchLearn (a_0, lines 12-21) has more complicated analysis subroutines than PRECISION (a_1, lines 31-34): (i) computing bit-level counter distributions; (ii) extracting large flows; (iii) removing extracted flows from multilevel sketch, and (iv) checking the termination condition. The termination depends on the Gaussian distribution of counter-distributions where large flows are extracted. It extracts large flows from the multilevel sketch, with flow key, estimated flow size, and accuracy loss. For its part, PRECISION directly obtains the flow key and corresponding flow size.

Algorithm 4 Control Plane Transaction

```
   Input: Packet p with iKey
1  initialization: skl[r][c][K] ← {0}; pre[M][N] ← {0};
   large flow list F ← ∅;
2  Function r0(skl):
3      for( i=0; i<r; i++ )do
4          for( j=0; j<c; j++ )do
5              for( k=0; k<K; k++ )do
6                  skl[i][j][k] ← read_reg_k(i * c + j);
7  end
8  Function c0():
9      for( k=0; k<K; k++ )do
10         clear_reg_k();
11 end
12 Function a0(skl):
13     θ ← 1/2; F ← ∅;
14     while true do
15         bit-level distributions
           {N(p[k], σ²[k])} = DIST(skl);
16         F ← F ∪
           extract_large_flow(θ, skl, {N(p[k], σ²[k])});
17         remove(skl, F);
18         if F == ∅ or terminate({N(p[k], σ²[k])})
           then
19             break;
20     end
21 end
22 Function r1(pre):
23     for( i=0; i<M; i++ )do
24         for( j=0; j<N; j++ )do
25             pre[i][j] ← read_reg_i(j);
26 end
27 Function c1():
28     for( i=0; i<M; i++ )do
29         clear_reg_i();
30 end
31 Function a1(pre):
32     for( i=0; i<M; i++ )do
33         sort(pre[i]);
34 end
35 while true do
36     tmp ← read_flag(0);
37     if tmp == 0 then
38         r1(); c1(); a1();
39     else if tmp == 1 then
40         r0(); c0(); a0();
41     wait until reverse time;
42     write_flag(^tmp);
43 end
```

The control plane coordinates with the PDP through the flag register to achieve an interleaved sketch. Figure 7 presents the transactions in time sequence. We denote the SketchLearn pipeline with $pipeline_0$ and PRECISION

pipeline with $pipeline_1$. Initially, the flag register is 0. The PDP accesses $pipeline_0$ until time t_1, at which point the control plane reaches timeout and reverses the flag register. The PDP then works on $pipeline_1$, since the flag register is 1 (time t_1-t_2); during this time, the control plane reads and clears the registers of $pipeline_0$. The control plane begins to analyze the statistics once the read routine has been completed. The control plane reverses the flag once the clearing routine has been completed and timeout has been reached (time t_2). Thus, the PDP and control plane exchange working pipelines (time t_2-t_3).

E. NETWORK-WIDE QUERY

1) NORMALIZING FREQUENCY

The control plane gets the potential large flows of all epochs. However, the time intervals of epochs are not necessarily equal. We use the normalized frequency fre_{ij} (1) rather than the absolute flow size f_{ij} to quantify the j^{th} flow in epoch t_i. The flow is identified by key_{ij}. The epoch t_i is within the time interval $[t_{i_{start}},\ t_{i_{end}})$.

$$fre_{ij} = \frac{f_{ij}}{t_{i_{end}} - t_{i_{start}}} \tag{1}$$

Additionally, we report the total flow size T_i (2) of epoch t_i

$$T_i = \sum_{j=1}^{|l_i|} f_{ij} \tag{2}$$

2) QUERY MECHANISM

Given that the normalized frequency resides in distributed entities, how can a network-wide query be achieved? We present two mechanisms: active query and passive query. In the active query, the control planes report flows whose frequency exceeds a certain threshold to the management plane by themselves. We encapsulate the potential large flow keys and corresponding frequencies in a query packet payload, which is sent to the management plane. In the passive query, moreover, the management plane sends a query request to selected control planes, after which the control planes report the results.

Figure 8 outlines the packet format of the query. To facilitate cooperation with existing protocols, we insert the query header into the L4 layer as an application layer protocol. We choose TCP for the active query and UDP for the passive query, which is consistent with the industrial standard [32]. A particular port is reserved for query packets so that the management plane can distinguish these packets from regular packets. The management plane reports only when this particular port is detected. The control planes also have reserved ports in the same way that the management plane does.

There are four major fields in the query protocol: *OP*, *TS*, *FR*, and *SEQ*. *OP* represents the operation of a query; this can be *Get*, *SET*, *Delete*, or any other type that is possible to use in the management plane. *TS* stores the timestamp of the epoch specified by this query. *FR* indicates the fraction

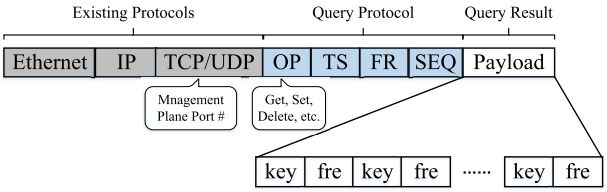


FIGURE 8. Network protocol of query.

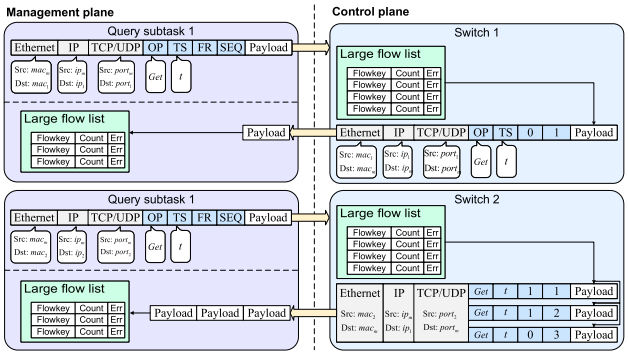


FIGURE 9. Active query example.

of records contained in one timestamp. *FR* is 0 for a single packet or the last packet, and 1 otherwise. *SEQ* is a sequence number used to ensure the reliable transmission of a single query.

Query packet forwarding is realized using existing routing protocols. The Ethernet, IP, and TCP/UDP headers of a query are set at the management plane or control plane level according to the data directed to the query. For active queries, the management plane may reset the control plane statistics in order to remove records and make space for new records.

For example, in Figure 9, assume that the management plane (mac_m, ip_m, $port_m$) queries the records of epoch t in $switch_1$ (mac_1, ip_1, $port_1$) and $switch_2$ (mac_2, ip_2, $port_2$). The management plane generates two packets and sets the Ethernet, IP, and UDP headers accordingly. In the query header, *OP* is get, *TS* is t, and *FR* and *SEQ* are unused. The payload is vacant. The control planes of the two switches receive a query packet from ports $port_1$ and $port_2$, respectively and parse the packet to determine whether it contains query requests from the management plane. $switch_1$ sets the Ethernet, IP, and TCP headers, sets *FR* to 0 and *SEQ* to 1, encapsulates the large flows of epoch t in the payload, and sends the packet. $switch_1$ fragments the records into three packets, such that *SEQ* is distributed among among 1, 2 and 3.

3) STATISTICS

The management plane gets the large flow list l_i of epoch i in per-flow frequency form $l_i = \{fre_{ij} \mid (key_{ij}, fre_{ij}) \in l_i, 1 \le j < |l_i|\}$ and the total flow size T_i. To find heavy hitters, the management plane extracts flows whose frequency exceeds a threshold. To find heavy changers (whose frequency changes sharply in two consecutive epochs t_1, t_2, assuming that l_{t_1} is the large flow list of t_1 and l_{t_2} is the

large flow list of t_2), we choose a flow $f_i \in l_{t_1}$ ($f_j \in l_{t_2}$) and compute the frequency change in $f_i \in l_{t_2}$ ($f_j \in l_{t_1}$). If $|fre_{i1} - fre_{i2}| > threshold$ ($|fre_{j1} - fre_{j2}| > threshold$), we identify f_i (f_j) as a heavy changer. We restore the per-flow size of list l_i by (3):

$$f_{ij} = fre_{ij} \cdot T_i \tag{3}$$

With the per-flow frequency distribution, many other statistics (such as entropy) can also be computed.

4) OVERHEAD

The interleaved sketch not only supports network-wide queries, but also provides a low report overhead. By reporting only large flows instead of all statistics to the management plane, we reduce the overhead by several orders of magnitude. For example, SketchLearn uses $r \cdot c \cdot (K+1)$ registers. Assuming $r = 1, c = 256, K = 32$. In our approach, the payload size is only related to the cardinality of the large flow set. Usually, the cardinality of large flows is no more than 3000 per second, while the payload is only $3000 \cdot (4+8) = 35$ KB and can be encapsulated in one packet. The centralized controller can read a maximum of K register slots at a time through INT. It will need 256 packets to fetch all of the raw statistics. We thus reduce the number of packets by 256×.

IV. EVALUATION

A. EXPERIMENTAL SETUP

1) TESTBED

We build a P4 prototype of the interleaved sketch with different combinations of sketch algorithms. We test all P4 programs on a Barefoot Tofino commodity programmable switch with a 3.2 Tbps switching chip and an Intel Pentium D-1517 CPU (1.6GHz, 4 cores). We connect the switch with two DELL PowerEdge R820 Servers, using two 40Gbps QSFP links. Each server has an Intel XL710-QDA2 NIC with two 40Gbps ports, two Intel Xeon E5-4603 CPUs and 256GB memory. We install MoonGen [33], a high-speed packet generator built on Lua and DPDK [34], to send and receive packets on the servers, which can achieve a stable speed of 38.04 Gbps. We adopt a DELL PC as the management plane to query statistics from Tofino.

2) TRACES

We test our prototype using CAIDA-2018, the Anonymized Internet Trace 2018 (hereafter denoted as CAIDA), which is collected from the Equinix-Chicago ISP backbone link. We use a one-hour long CAIDA trace that contains a mix of UDP, TCP, and other packets. We filter out the UDP and TCP packets to generate our test workloads.

3) METHODOLOGY

One server sends real-time traffic to Tofino through the 40GbE link. It reads all workloads into memory and emits the flows according to the timestamps in trace files. The data rate follows the trace file, thanks to the MoonGen framework, which eliminates kernel-space overhead utilizing DPDK. Tofino PDP measures the traffic with the interleaved sketch and forwards the traffic to the other server. The Tofino control plane executes a Python script to invoke the register operation APIs that are exposed over Thrift-RPC, which read and clear the PDP registers. The control plane also encapsulates the large flow list into query packets and reports them to the management plane machine, through a dedicated 1GbE.

4) METRICS

- RE (Relative Error): $\frac{f_i - \hat{f}_i}{f_i}$, f_i is the actual flow size and $\hat{f}_i$ is the estimated flow size.
- ARE (Average Relative Error): $\frac{1}{n}\sum_{i=1}^{n}\frac{f_i - \hat{f}_i}{f_i}$, n is the flow cardinality
- MSE (Mean Square Error): $\frac{1}{n-1}\sum_{i=1}^{n}(f_i - \bar{f})^2$, where $\bar{f} = \frac{1}{n}(f_1 + f_2 + \cdots + f_n)$
- F_1 score: $\frac{2}{F_1} = \frac{1}{P} + \frac{1}{R}$.

$$F_1 = \frac{2 \cdot P \cdot R}{P + R} \tag{4}$$

Here, Precision Rate (P) is the ratio of true positives (TP) among true positives and false positives (FP),

$$P = \frac{TP}{TP + FP} \tag{5}$$

and Recall Rate (R) is the ratio of true positives among true positives and false negatives (FN).

$$P = \frac{TP}{TP + FN} \tag{6}$$

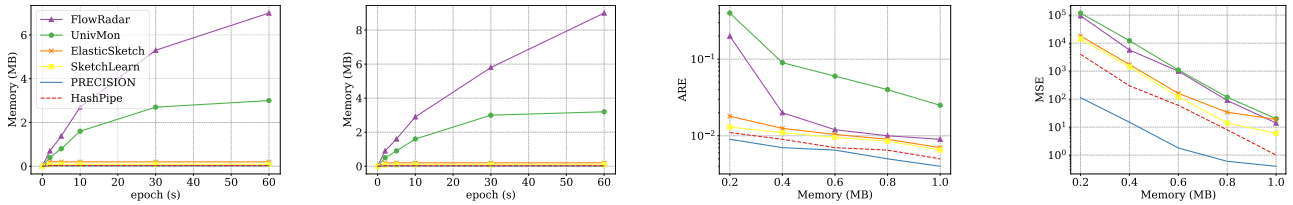
B. PDP

We implement the algorithms on Tofino ASIC, except for SpaceSaving and RAP, which cannot feasibly be used on the ASIC.

1) TUNING PARAMETERS

We tune the parameters of these algorithms, i.e. epoch size and memory usage, of these algorithms. Figure 10(a)-(b) show the memory usage under different epoch sizes to achieve 0.99 F_1 score for HH and HC detection respectively. As epoch size increases, FlowRadar and UnivMon use more memory space, while the others remain in a state of low memory usage. In Figure 10(b), when the epoch length is 60 seconds, FlowRadar uses 8.7 MB memory, which is 2.3× that of UnivMon and 350× that of PRECISION. We set the epoch length to two seconds, as all algorithms use a relatively small amount of memory space under this epoch size, and the interval is fine-grained to enable reporting of large flows.

Figure 10(c)-(d) show the accuracy of these algorithms under different memory sizes when the epoch length is 2 seconds. Overall, more memory space makes the algorithms more accurate. However, this benefit increases slowly for PRECISION, HashPipe, and SketchLearn, as they are already sufficiently accurate with 200 KB memory space.

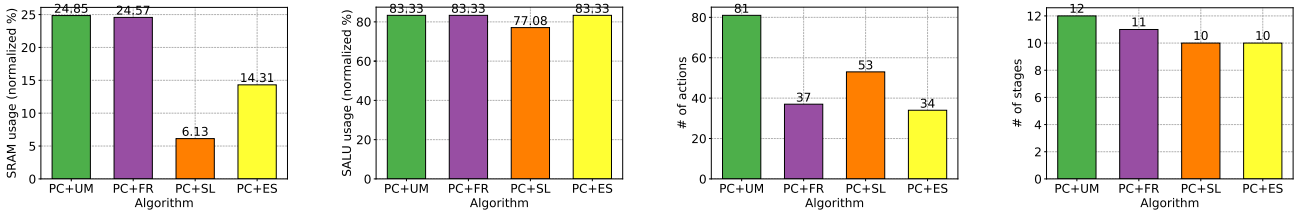


(a) HH memory (b) HC memory (c) HH ARE (d) HH MSE

FIGURE 10. Tune parameters.

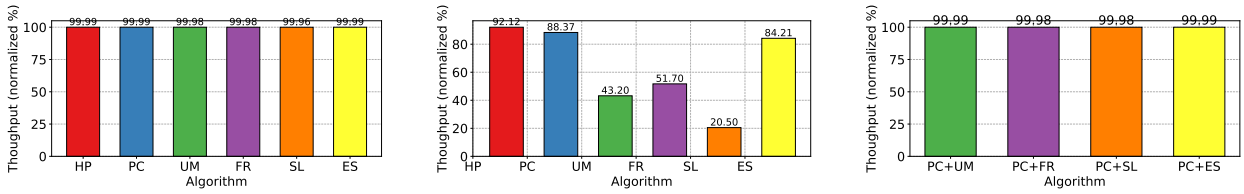
TABLE 2. Resource usage on Tofino ASIC.

Resource	Switch.p4	HashPipe	PRECISION	UnivMon	FlowRadar	SketchLearn	ElasticSketch
Match Crossbar	50.13%	2.51%	1.27%	2.13%	2.05%	1.89%	5.9%
Hash Bits	32.35%	3.87%	3.64%	3.81%	4.17%	3.79%	2.3%
SRAM	29.79%	2.19%	1.81%	23.04%	22.76%	4.32%	12.5%
TCAM	28.47%	0%	0%	0%	0%	0%	0%
VLIW Actions	34.64%	2.34%	2.21%	5.31%	4.72%	2.86%	5.52%
Stateful ALUs	15.63%	16.67%	8.33%	75.00%	75.00%	68.75%	75.00%
Stages	12	5	4	12	11	10	10



(a) SRAM resource usage (b) SALU resource usage (c) # of actions (d) # of stages

FIGURE 11. Interleaved sketch resource usage on Tofino ASIC.



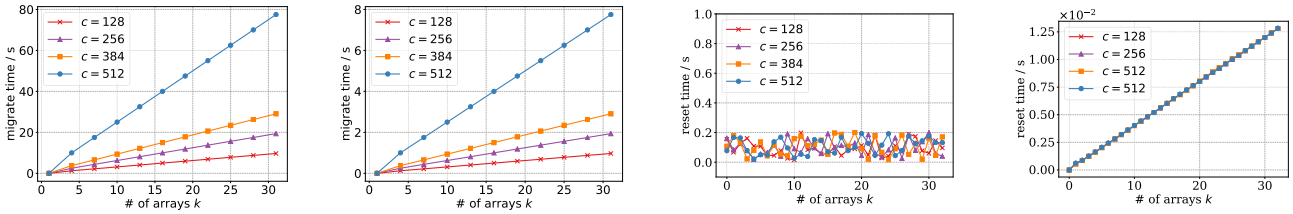
(a) ASCI throughpt: single algorithm (b) Software throughpt: SketchVisor (c) ASCI throughpt: Interleaved sketch

FIGURE 12. Throughput of single algorithm and interleaved sketch.

2) RESOURCE USAGE

We implement the algorithms using tuned parameters. Table 2 presents the percentage of consumed resources over all resources on BareFoot Tofino ASIC, where switch.p4 is a P4 prototype that implements the typical functions (e.g., L2/L3 forwarding, VLAN, and QoS) of a switch.

HashPipe and PRECISION are lightweight algorithms that use minimal hardware resources. For example, PRECISION uses only 1.81% SRAM with 8.33% SALU to operate them to track large flows and accomplishes all PDP transactions in 4 stages. While UnivMon, FlowRadar, SketchLearn, and ElasticSketch are regular sketches, that consume a significant amount of resources; they all need over 65% SALUs, since they have more register arrays. For example, ElasticSketch costs 12.5% SRAM and 75% SALU, which are 6.9× and 9× higher than that of PRECISION. Moreover, they all require no less than ten stages to accomplish PDP transactions. These algorithms overlap with switch.p4 so that they can be arranged in the Tofino pipeline. SketchLearn SRAM usage is also relatively low compared with other regular algorithms, but still needs twice as much SRAM as HP and PC.



(a) INT migrate time (b) control plane migrate time (c) INT reset time (d) control plane reset time

FIGURE 13. Migrate and reset time.

It would be unrealistic to use two regular sketch algorithms in Tofino ASIC to resource restrictions. Accordingly, the only solution is to combine a regular sketch algorithm with a lightweight one in the interleaved sketch. We choose PRECISION as the lightweight part in the following experiments, as it is more accurate and resource-efficient than HashPipe. We test the effectiveness of different sketch algorithms in acting as the regular part in the interleaved sketch.

Figure 11 shows the combined resource usage. The SRAM resource usage is the sum of the resource usage of two algorithms, as the SRAM is allocated as a whole block to each algorithm, such that the unused slots of the block cannot be assigned to other algorithms. The SRAM usage is between 6.13% to 24.85%. Taking the switch.p4 into account, the overall SRAM usage is under 55%; the remaining SRAM space is large enough to accommodate other functions. The SALU usage is also the sum of the usage of two algorithms, as they cannot be multiplexed. The number of actions is slightly lower than the sum of two algorithms, as some of the actions are collective and shared. The number of stages remains consistent with those of the regular sketch, as the lightweight sketch is distributed across different physical stages and executed in parallel.

3) THROUGHPUT

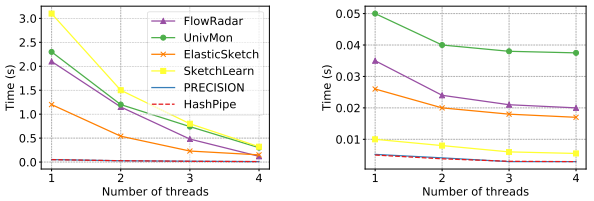
We stress-test the throughput of the sketch algorithms on Tofino and compare it with that of SketchVisor, the software packet processing platform. Figure 12 shows the normalized throughput to the line-rate speed without measurement. Tofino realizes line-rate processing for different algorithms (Figure 12(a)). While SketchVisor is not stable for computationally intensive algorithms (Figure 12(b)) such as UnivMon, FlowRadar and SketchLearn.

Figure 12(c) presents the results of the stress test of interleaved sketch. The processing speed is preserved and the variance is minimal. The high performance is a result of the fact that two sketch pipelines have no dependency.

C. CONTROL PLANE

1) REGISTER COLLECTING AND RESETING

Figure 13 shows the register operation time on SketchLearn. Figure 13(a) shows the register migration time through INT. The migration time increase sharply when the c and k increase, as more INT packets are required. Figure 13(b) shows control plane migration time, which is faster than INT. For our interleaved sketch, the migration spends less than 1 second ($k = 32$, $c = 128$, $r = 1$), which is suitable for our interval. Figure 13(c) shows that INT has an undetermined reset time. Figure 13(d) shows the control plane reset time, which increases linearly as the number of register arrays increases. The reset time has is not related to the register array size, as each register array is reset as a whole.



(a) Analyzing time (b) Query latency

FIGURE 14. Analyzing and query time.

2) STATISTICAL ANALYSIS

Figure 14(a) illustrates the control plane analysis time for different algorithms. Four threads are sufficient to accomplish all analysis in 0.3 seconds. The analysis begins after register reading and is conducted in parallel with register clearing. The analysis will end before the next reading ends so that the analysis processes do not conflict.

3) QUERY LATENCY

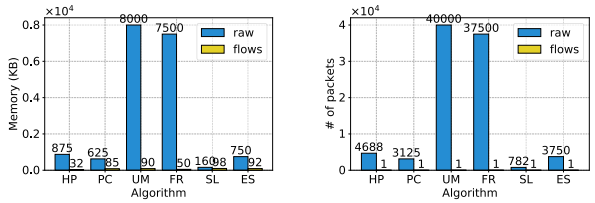
Figure 14(b) illustrates the query latency. The time between 0.001-0.05, depending on the size of the statistics.

The distributed control planes analyze statistics in parallel and are faster than the centralized controller, which analyzes statistics in a serial manner. Due to the reporting of only large flows, the total report latency in the interleaved sketch is less than 1.35 seconds, while the centralized controller takes more than 3 seconds to transmit SketchLearn raw statistics in our experiment.

D. MANAGEMENT PLANE

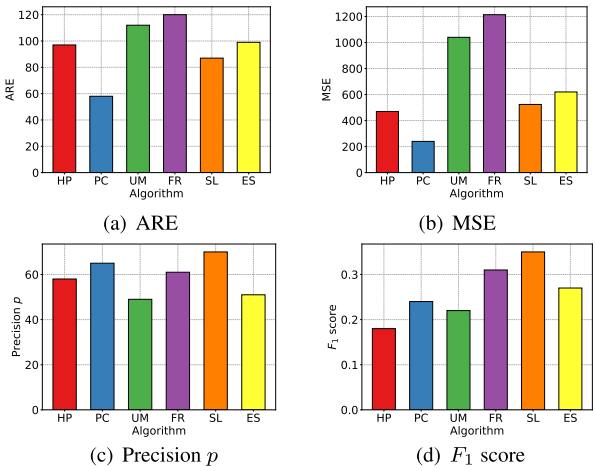
1) REPORT OVERHEAD

Figure 15(a) is the report memory size (bandwidth) for a single switch. This size includes the overhead of packet headers. The interleaved sketch reports only large flows, thereby saving 38.8%-99.3% report memory (bandwidth) relative to



(a) Communication memory size (b) # of packets

FIGURE 15. Communication overhead.



(a) ARE (b) MSE

(c) Precision p (d) F_1 score

FIGURE 16. Accuracy without interleaved sketch.

INT, which transmits all raw statistics to the management plane. Figure 15(b) shows the number of report packets. The interleaved sketch reports large flows with a payload of less than 50KB per packet, while INT uses 782-40000 packets to report raw statistics.

The overhead for multiple switches in a network is far lower than that for the centralized controller, as each switch only generates a single small packet. The proportion between raw statistics size and large flow size in Figure 15 still holds.

2) ACCURACY

We first measure the accuracy of the algorithms without interleaved sketch for HH detection. We then measure the accuracy of the interleaved sketch for different applications.

Figure 16 shows the accuracy of the algorithms without the interleaved sketch. The ARE and MSE are quite large, while the precision and F_1 score are inadequate. The error arises as a result of the inconsistency of statistics.

Figure 17(a)-(c) shows the interleaved sketch accuracy for Heavy Hitter detection. The ARE is below 0.03, MSE is below 17, and F_1 score is above 0.97 for all algorithms. For Heavy Change detection (Figure 17(d)-(f)), the ARE is below 0.06, MSE is below 22, and F_1 score is above 0.94. For all algorithms, the interleaved sketch achieves the ideal accuracy for HH and HC detection.

Figure 17(g) shows the relative error for cardinality estimation. SketchLearn, as the regular sketch, produces the lowest error at 4%. Its sketch structure is a better fit for large cardinality. All relative errors for cardinality estimation are below 9%.

Figure 17(h) shows the relative error for entropy estimation. SketchLearn outperforms other sketch algorithms for entropy estimation. All errors are below 12%. The cardinality and entropy estimation are also close to ideal.

In short, the interleaved sketch provides consistent statistics, thus providing accurate results for applications.

V. RELATED WORKS

INT. Kim *et al.* [35] built the first INT prototype on software programmable switches. Each switch uses INT to periodically push telemetry packets to the end host. The packets provide near-instantaneous observation of the network status. PacketHistory [36] and Planck [37] mirror traffic for further analysis, which incurs high bandwidth. Everflow [38] only traces specific packets by implementing a "match and mirror" policy on commodity switches, thereby reducing the bandwidth.

INT is efficient for fetching a small number of PDP states. The interleaved sketch can be improved by reducing the latency and bandwidth for fetching massive states in INT.

A. QUERY LANGUAGES

Query languages [39], [40] can express sophisticated measurement requirements. They specify patterns to monitor specific flows and query the results.

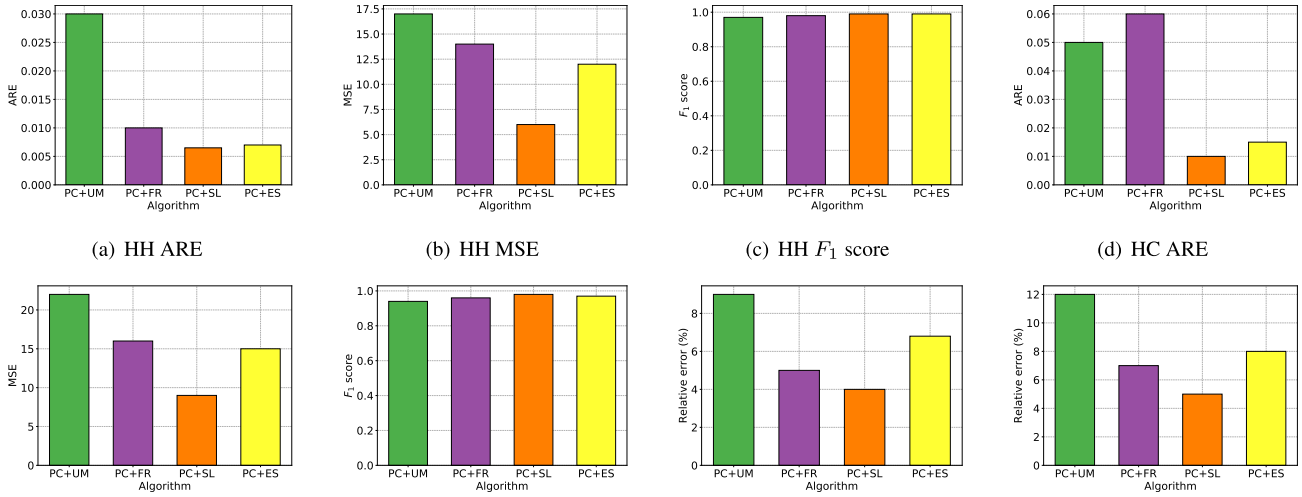
We design a preliminary query protocol in our interleaved sketch to read PDP statistics. We could potentially add more query types and telemetry tasks to expand the interleaved sketch.

B. TIME SYNCHRONIZATION

Consistency has been widely studied in distributed computing research, providing consistency guarantees for network telemetry. Strong consistency systems [41] align epoch boundaries and ensure that every packet is observed by all entities at the same epoch. Weak consistency systems, e.g., Lamport Clock [42] and Distributed Snapshots [43], synchronize per-packet epochs in a best-effort manner. Each entity renews its time with reference to a local clock and embeds the current epoch in every outgoing packet. Accordingly, the epochs among entities may diverge, and a packet may be monitored at different epochs. By contrast, the interleaved sketch can achieve consistency through network-wide epoch synchronization, with the primary objective of all entities being synchronized at the same epoch.

C. PROGRAMMING FRAMEWORKS

Sluice [23] and P4HLPc [25] aim to hide the underlying restrictions of programmable switches in a network in order to provide unified high-level programming for all programmable switches. SNAP [24] provides a centralized programming model that views the distributed switches as centralized, relieving programmers of the need to place and optimize access to switch resources.



(a) HH ARE (b) HH MSE (c) HH F_1 score (d) HC ARE

(e) HC MSE (f) HC F_1 score (g) Cardinality estimation (h) Entropy estimation

FIGURE 17. **Interleaved sketch accuracy.**

VI. CONCLUSION

This paper investigates the inconsistency problem in conventional network telemetry systems. These systems rely on interval methods that track flow statistics in each epoch and collect statistics at the end of each epoch. The inconsistency derives from the joint operation of the PDP and the control plane, which modify statistics simultaneously.

We present our solution, the interleaved sketch, in which the PDP and the control plane access two sketch pipelines in an interleaved way. The PDP and the control plane communicate through a flag register, which indicates the pipeline access order. Rather than duplicating the pipeline, we use asymmetric sketch algorithms, i.e. a regular sketch algorithm and a lightweight sketch algorithm. This asymmetric sketch approach trades off resource consumption and accuracy. We further propose the decentralized controller, which decouples the control plane and management plane. Each switch is self-supervised by the interplay of its control plane and PDP.

Experimental results show that our interleaved sketch achieves line-rate processing on the Tofino switch. Moreover, our solution improves the F_1 score from 0.3 to 0.97 for HH detection, and also performs well on HC detection, cardinality estimation, and entropy estimation, with only a 6% rise in resource consumption. The decentralized controller lowers the latency and saves bandwidth for transmission, meaning that the telemetry system can detect anomalies in a timely fashion.

In conclusion, the interleaved sketch approach enables telemetry systems with line-rate processing, high accuracy, and low latency on a Tofino switch. Our work may provide valuable guidance for future telemetry system design.

REFERENCES

[1] T. Benson, A. Anand, A. Akella, and M. Zhang, "MicroTE: Fine grained traffic engineering for data centers," in *Proc. 7th Conf. Emerg. Netw. Exp. Technol.*, 2011, p. 8.

[2] A. Kabbani, M. Alizadeh, M. Yasuda, R. Pan, and B. Prabhakar, "AF-QCN: Approximate fairness with quantized congestion notification for multi-tenanted data centers," in *Proc. 18th IEEE Symp. High Perform. Interconnects*, Aug. 2010, pp. 58–65.

[3] S. Dong, K. Abbas, and R. Jain, "A survey on distributed denial of service (DDoS) attacks in SDN and cloud computing environments," *IEEE Access*, vol. 7, pp. 80813–80828, 2019.

[4] A. Metwally, D. Agrawal, and A. El Abbadi, "Efficient computation of frequent and top-*k* elements in data streams," in *Proc. Int. Conf. Database Theory*. Berlin, Germany: Springer-Verlag, 2005, pp. 398–412. [Online]. Available: https://www.springer.com/gp/book/9783540242888

[5] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford, "Heavy-hitter detection entirely in the data plane," 2016, *arXiv:1611.04825*. [Online]. Available: https://arxiv.org/abs/1611.04825

[6] Cisco. (2019). *Cisco IOS NetFlow*. [Online]. Available: https://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html

[7] A. A. Abbasi, A. Abbasi, S. Shamshirband, A. T. Chronopoulos, V. Persico, and A. Pescapè, "Software-defined cloud computing: A systematic review on latest trends and developments," *IEEE Access*, vol. 7, pp. 93294–93314, 2019.

[8] ONF. (2019). *Software-Defined Networking (SDN) Definition*. [Online]. Available: https://www.opennetworking.org/sdn-definition/?nab=1&utm_referrer=https%3A%2F%2Fwww.google.com%2F

[9] Y. Li, R. Miao, C. Kim, and M. Yu, "FlowRadar: A better NetFlow for data centers," in *Proc. 13th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2016, pp. 311–324.

[10] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman, "One sketch to rule them all: Rethinking network flow monitoring with UnivMon," in *Proc. ACM SIGCOMM Conf.*, 2016, pp. 101–114.

[11] Q. Huang, P. P. C. Lee, and Y. Bao, "Sketchlearn: Relieving user burdens in approximate measurement with automated statistical inference," in *Proc. Conf. ACM Special Interest Group Data Commun.*, 2018, pp. 576–590.

[12] 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. Conf. ACM Special Interest Group Data Commun.*, 2018, pp. 561–575.

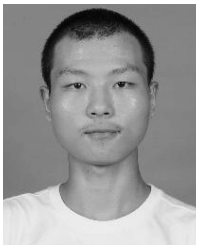
[13] Barefoot. (2019). *Barefoot Tofino: World's Fastest P4-Programmable Ethernet Switch ASICs*. [Online]. Available: https://www.barefootnetworks.com/products/brief-tofino/

[14] P4 Language Consortium. (2019). *P4*. [Online]. Available: https://p4.org/

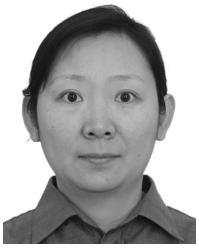
[15] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 99–110, Oct. 2013.

[16] Wikipedia. (2019). *Domain–Specific Language*. [Online]. Available: https://en.wikipedia.org/wiki/Domain-specific_language

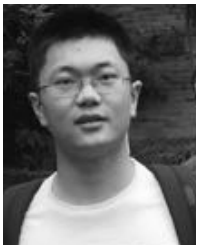
[17] Intel. (2019). *Intel Flexpipe*. [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/ethernet-switch-fm6000-series-brief.pdf
[18] DELL. (2019). *Dell EMC Networking S5148F-ON Series Switch*. [Online]. Available: https://i.dell.com/sites/csdocuments/Shared-Content_data-Sheets_Documents/en/dell-emc-networking-s5148f-on-spec-sheet.pdf
[19] Cisco. (2019). *Cisco Data Center Switches*. [Online]. Available: https://www.cisco.com/c/en/us/products/switches/data-center-switches/index.html
[20] HUAWEI. (2019). *Huawei Data Center Switches*. [Online]. Available: https://e.huawei.com/uk/products/enterprise-networking/switches/data-center-switches
[21] S. Muthukrishnan, "Data streams: Algorithms and applications," *Found. Trends Theor. Comput. Sci.*, vol. 1, no. 2, pp. 117–236, 2005.
[22] J. N. Dieffenderfer and R. N. Kalla, "Ping-pong data buffer for transferring data from one data bus to another data bus," U.S. Patent 5 224 213, Jun. 29, 1993.
[23] V. Natesh, P. G. Kannan, A. Sivaraman, and R. Netravali, "Sluice: Network-wide data plane programming," in *Proc. ACM SIGCOMM Conf. Posters Demos*, 2019, pp. 156–158.
[24] M. T. Arashloo, Y. Koral, M. Greenberg, J. Rexford, and D. Walker, "SNAP: Stateful network-wide abstractions for packet processing," in *Proc. ACM SIGCOMM Conf.*, 2016, pp. 29–43.
[25] Z. Hang, M. Wen, Y. Shi, and C. Zhang, "Programming protocol-independent packet processors high-level programming (P4HLP): Towards unified high-level programming for a commodity programmable switch," *Electronics*, vol. 8, no. 9, p. 958, 2019.
[26] P4 Language Consortium. (2019). *P4-14 Specification*. [Online]. Available: https://p4lang.github.io/p4-spec/p4-14/v1.0.5/tex/p4.pdf
[27] R. Ben Basat, G. Einziger, R. Friedman, and Y. Kassner, "Randomized admission policy for efficient top-*k* and frequency estimation," in *Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM)*, May 2017, pp. 1–9.
[28] R. Ben-Basat, X. Chen, G. Einziger, and O. Rottenstreich, "Efficient measurement on programmable switches using probabilistic recirculation," in *Proc. IEEE 26th Int. Conf. Netw. Protocols (ICNP)*, Sep. 2018, pp. 313–323.
[29] P4 Group. (2019). *Specification for the P4 Runtime Control-Plane API*. [Online]. Available: https://github.com/p4lang/p4runtime/
[30] P4 Group. (2019). *Packet Test Framework*. [Online]. Available: https://github.com/p4lang/ptf/
[31] Apache. (2019). *Apache Thrift*. [Online]. Available: https://github.com/apache/thrift/
[32] R. Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, H. C. Li, R. McElroy, M. Paleczny, D. Peek, and P. Saab, "Scaling memcache at Facebook," in *Proc. NSDI*, vol. 13, 2013, pp. 385–398.
[33] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle, "MoonGen: A scriptable high-speed packet generator," in *Proc. Internet Meas. Conf.*, 2015, pp. 275–287.
[34] Libmoon. (2019). *DPDK is the Data Plane Development Kit That Consists of Libraries to Accelerate Packet Processing Workloads Running on a Wide Variety of CPU Architectures.* [Online]. Available: https://www.dpdk.org/
[35] C. Kim, A. Sivaraman, N. Katta, A. Bas, A. Dixit, and L. J. Wobker, "In-band network telemetry via programmable dataplanes," in *Proc. ACM SIGCOMM*, 2015, pp. 1–2.
[36] N. Handigol, B. Heller, V. Jeyakumar, D. Mazières, and N. McKeown, "I know what your packet did last hop: Using packet histories to troubleshoot networks," in *Proc. 11th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2014, pp. 71–85.
[37] J. Rasley, B. Stephens, C. Dixon, E. Rozner, W. Felter, K. Agarwal, J. Carter, and R. Fonseca, "Planck: Millisecond-scale monitoring and control for commodity networks." *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 407–418, 2015.
[38] Y. Zhu, N. Kang, J. Cao, A. Greenberg, G. Lu, R. Mahajan, D. Maltz, L. Yuan, M. Zhang, B. Y. Zhao, and H. Zheng, "Packet-level telemetry in large datacenter networks," in *Proc. SIGCOMM*, Aug. 2015, pp. 479–491. [Online]. Available: https://www.microsoft.com/en-us/research/publication/packet-level-telemetry-in-large-datacenter-networks/
[39] A. Gupta, R. Birkner, M. Canini, N. Feamster, C. Mac-Stoker, and W. Willinger, "Network monitoring as a streaming analytics problem," in *Proc. 15th ACM Workshop Hot Topics Netw.*, 2016, pp. 106–112.
[40] O. Tilmans, T. Bühler, I. Poese, S. Vissicchio, and L. Vanbever, "Stroboscope: Declarative network monitoring on a budget," in *Proc. 15th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2018, pp. 467–482.
[41] L. Lamport, "Paxos made simple," *ACM SIGACT News*, vol. 32, no. 4, pp. 18–25, 2001.
[42] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM*, vol. 21, no. 7, pp. 558–565, Jul. 1978.
[43] N. Yaseen, J. Sonchack, and V. Liu, "Synchronized network snapshots," in *Proc. Conf. ACM Special Interest Group Data Commun.*, 2018, pp. 402–416.



ZIJUN HANG received the B.S. degree from the Beijing Institute of Technology, Beijing, China, in 2017. He is currently pursuing the M.S. degree with the College of Computer, National University of Defense Technology. His research interests include SDN, INT, resource management, and workload scheduling.



MEI WEN received the B.S., M.S., and Ph.D. degrees in computer science and technology from the National University of Defense Technology, China, in 1995, 1999 and 2006, respectively, where she is currently a Professor with the Computer College. Her research interests include computer architecture, parallel programming, and scientific computing.



YANG SHI received the B.S. and M.S. degrees from the National University of Defense Technology, Changsha, China, in 2014 and 2016, respectively, where he is currently pursuing the Ph.D. degree with the College of Computer. His research interests include distributed and parallel computing, resource management, and workload scheduling.

CHUNYUAN ZHANG received the B.S., M.S., and Ph.D. degrees in computer science and technology from the National University of Defense Technology, China, in 1985, 1990, and 1996, respectively, where he is currently a Professor with the Computer College. He is also the Director of a series of research projects, including National Natural Science Foundation projects of China. His research interests include computer architecture, parallel programming, embedded systems, and scientific computing.