A Robust Counting Sketch for Data Plane Intrusion Detection

Sian Kim[†], Changhun Jung[†], Rhongho Jang[*], David Mohaisen[‡] and DaeHun Nyang[†]
[†]Ewha Womans University, [*]Wayne State University, [‡]University of Central Florida
[†]{ksy60a, mizno, nyang}@ewha.ac.kr, [*]r.jang@wayne.edu, [‡]David.Mohaisen@ucf.edu

Abstract— **Demands are increasing to measure per-flow statistics in the data plane of high-speed switches. However, the resource constraint of the data plane is the biggest challenge. Although existing in-data plane solutions improve memory efficiency by accommodating Zipfian distribution of network traffic, they cannot adapt to various flow size distributions due to their static data structure. In other words, they cannot provide robust flow measurement under complex traffic patterns (e.g., under attacks). Recent works suggest dynamic data structure management schemes, but the high complexity is the major obstruction for the data plane deployment. In this paper, we present Count-Less (CL) sketch that enables robust and accurate network measurement under a wide variety of traffic distributions without dynamic data structure updates. Count-Less adopts a novel sketch update strategy, called *minimum update* (CL-MU), which approximates the conservative update strategy of Count-Min for fitting into in-network switches. Not only theoretical proof on CL-MU's estimation but also comprehensive experimental results are presented in terms of estimation accuracy and throughput of CL-MU, compared to Count-Min (baseline), Elastic sketch, and FCM sketch. More specifically, experiment results on security applications including estimation errors under various skewness parameters are provided. CL-MU is much more accurate in all measurement tasks than Count-Min and outperforms FCM sketch and Elastic sketch, state-of-the-art algorithms without the help of any special hardware like TCAM. To prove its feasibility in the data plane of a high-speed switch, CL-MU prototype on an ASIC-based programmable switch (Tofino) is implemented in P4 language and evaluated. In terms of data plane latency, CL-MU is faster than FCM, while consuming fewer resources such as hash bits, SRAM, and ALU of a programmable switch.**

I. Introduction

To detect malicious flows in networks, per-flow measurement is essential. The network flow measurement can be done either at a gateway or in the network. The gateway approach, like network function virtualization, gives more flexibility and scalability. However, the operational cost is high [39]. Recently, the networking community has been moving towards in-network intrusion detection with programmable switches to distribute the overhead and reduce the cost. Nevertheless, the in-network approach also faces several challenges, such as the constraints of hardware resources in switches, packet processing speed, and the distribution of the traffic to be measured. Especially, the traffic distribution is commonly accepted to follow Zipfian distribution, but in reality, it varies so widely across time and application type to show a variety of Zipf parameters.

[†]The first two authors contributed equally to this work.

Network and Distributed System Security (NDSS) Symposium 2023
27 February - 3 March 2023, San Diego, CA, USA
ISBN 1-891562-83-5
https://dx.doi.org/10.14722/ndss.2023.23102
www.ndss-symposium.org

Current in-network measurement solutions for anomaly detection fall into three categories: hardware-based, sampling-based, and sketch-based approaches. Hardware-based approaches take advantage of the advanced ternary content-addressable memory (TCAM) to perform non-delayed measurement [46], thanks to its parallel searching capability, but they have suffered from scalability issues owing to the scarcity of TCAM (i.e., tens of megabytes). Thus, sampling-based approaches become a viable option due to their simplicity and scalability. However, the traffic sampling approach is often criticized for the poor measurement accuracy or communication overhead with remote sample collectors [54]. Sketch-based approaches are yet another promising option for per-flow measurement [10], [11], [13], [15], [16], [27], [33], [34], [37], [40], [47], [59], [61], [63], [64], [66], [58]. In general, a sketch, as an approximate measurement solution, provides a good estimation accuracy under computation and memory constraints. Particularly, their lightweight encoding/decoding functions guarantee low-latency packet processing in a switch's data plane.

Among many interesting sketches, Count-Min sketch [10] is a simple yet powerful sketch that approximately counts a large volume of data stream using a small amount of memory. The simplicity of its operations (i.e., encoding and decoding) and data structure have led to its use for multiple security applications, such as DDoS, superspreader, heavy hitter, and heavy changer detections [19], [21], [25], [60], [62], [64], [65]. Moreover, several efforts have been made to improve its accuracy by applying different encoding strategies [12], [20], [51], [52]. On the other hand, the network community had a long-term observation on the Zipfian (Zipf) flow size distribution (FSD) of network traffic, thus motivated a series of works to tune the Count-Min's data structure for adapting the distribution [58], [62], [63], [66]. Intuitively, these sketches use a multi-layer filtering approach to break down and cascade flow measurement based on the flow size and in a certain order (e.g., mouse$\rightarrow$elephant or elephant$\rightarrow$mouse), as shown in Fig. 1. To adapt to the network FSD (i.e., Zipf), these approaches define more smaller-sized counters (e.g., 8-bit) to the mouse layer and assign fewer larger counters to the medium and elephant layers (i.e., pyramid-shaped data structure). The above data structure has to be tuned to a certain FSD by adjusting the number of counters in each layer. Our observation is that although this approach can improve the accuracy by assigning more counters to mouse flows (i.e., hash collision reduction and memory efficiency improvement), it cannot adapt to a sudden change of the FSD (caused by attack flows) since the data structure is fixed to be installed in the data plane before performing a measurement task. In general, the data structure is configured based on the historical record of the on-site FSD (i.e., usually benign network traffic). Unfortunately, given the broad spectrum of attack patterns, the approach cannot provide a robust flow measurement performance, in turn, a robust performance of intrusion detection. While a sketch can be configured to target a certain attack, it degrades the accuracy of the benign flow measurement and may result in false negative alarms (i.e., report benign flow as malicious). Therefore, it is crucial for a sketch to provide consistent accuracy across different FSDs.

In this paper, we answer an important question of how to design a robust sketch to embrace various Zipf distributions, including attack traffic, without dynamic adjustment of data structure. Especially, we aim to design a sketch that can robustly perform both benign and attack flow measurements, with all the implementation constraints of in-network devices (i.e., switches). Our sketch, called Count-Less, takes advantage of the pyramid-shaped data structure but proposes a novel update strategy, called the cross-layer counting with minimum update, or *minimum update* in short. With *minimum update*, Count-Less can flexibly utilize idle counters across all layers to guarantee a better flow survival rate, under various FSDs (see section V-C). Moreover, *minimum update* is a pipeline design approximation of the famous conservative update strategy of Count-Min, which improves the accuracy by minimizing redundant counter updates in our sketch. Last but not least, with the approximation, Count-Less is feasible in a switch's data plane for robust in-network flow measurement (see section VI for details). Through comprehensive experiments, Count-Less's approach is shown to be superior in terms of accuracy at a constant memory footprint and to be better in terms of data plane latency and packet processing throughput (see section V and VI). Our contributions are as follows:

1) We present Count-Less sketch that enables robust and accurate network flow measurement under both attack and normal traffic scenarios without dynamic adjustment of data structure. Count-Less applies a novel sketch update strategy, called *minimum update* (CL-MU), which approximates the conservative update strategy for fitting into in-network switches.
2) Comprehensive analyses, including theoretical proof on the error bound of CL-MU and experimental analysis, are presented. Extensive experiments confirm that CL-MU is superior to state-of-the-art schemes such as Elastic sketch [62] and FCM [62] in flow measurement and various security applications.
3) We verify CL-MU's feasibility by implementing a prototype in a programmable switch. The hardware version's performance is comparatively analyzed in terms of resource consumption and latency. We further reproduce the security applications in the data plane to verify their correctness.

Organization: The rest of the paper is organized as follows: we introduce our motivation, including arguments for and against existing approaches, and the design choice of Count-Less sketch in section II. In section III, we introduce our Count-Less sketch with the analysis in section IV. We evaluate Count-Less sketch in section V. We demonstrate our data plane implementation of Count-Less sketch and evaluate it in section VI. Lastly, we review the literature in section VII, and conclude in section VIII.

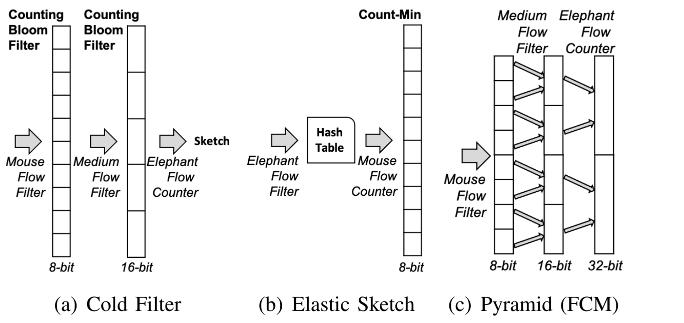


Fig. 1: Cascaded multi-sketch approaches with flow filters

II. MOTIVATING COUNT-LESS

Network traffic's Flow Size Distribution (FSD) is often skewed, following the Zipf's law. To date, there have been several efforts to address inefficient data structures designed for the skewed data streams [58], [62], [63], [66], as shown in Fig. 1. These approaches all fall in the same category, namely *multi-stage filtering*, a concept that was first introduced by Estan *et al.* [15]. Recent works cascade a number of sketches (i.e., shared counter/bit arrays) for a sequential flow filtering according to their size, thus we call them the cascaded multi-sketch approaches. To adapt to the Zipf FSD, these works preferentially assign more small counters to mouse flow filter and fewer large counters to medium and elephant filters.

As shown in Fig. 1(a), Cold Filter [66] concatenates two counting bloom filters with small counters (e.g., 8-bit) for filtering out mouse and medium flows sequentially, and finally stores elephants in an additional full-size (i.e., 32-bit per counter) data structure such as CM-CU, Space-Saving [41], FlowRadar [36]. On the contrary, Elastic sketch [62] counts and filters out elephant flows first using TCAM, then records mouse flows using the Count-Min sketch with 8-bit counters, as shown Fig. 1(b). For systematic filtering, Pyramid Sketch [63] presented a tree-based data structure that enlarges the size of each layer's counters as the decrements of the tree depth, as shown in Fig. 1(c). Recently, FCM sketch [58] extended a similar idea with a multi-tree design and implemented it in a programmable switch (state-of-the-art).

Intuitively, these approaches achieve better accuracy than Count-Min mainly because of their data structure design reflecting a skewness. Moreover, they can adapt to various FSDs by reconfiguring parameters like adjusting the number of counters or the number of layers, etc. For example, we might be able to reconfigure the data structure actively based on either historical records or attack detection results. However, the sketch configured with the historical records cannot deal with a sudden change of FSDs (e.g., attack traffic), and one with the attack detection suffers from a high latency caused by information gathering on the FSD of the ongoing attack traffic for proper parameter selection. Also, a large number of packets are missed while reconfiguring and installing sketches in the data plane. Due to those issues, a data structure for a sketch cannot be easily reconfigured on-the-fly without those downsides. In the following, we explain our observations of FSD's change and our approach to embracing these changes.

A. FSD Changes Over Time and Under Attacks

Although the aforementioned cascaded multi-sketch approaches can work well with certain FSDs, they cannot perform the robust network flow measurement for three reasons.

❶ FSD is unpredictable and varies over time, even for the same stream. Fig. 2 shows FSDs of 49 one-minute traces for each of UNSW [32] and CIC 2019 [1] datasets. Each grey line stands for the FSD of one-minute trace in the figure. As shown, FSDs can change a lot even for the same dataset over time.

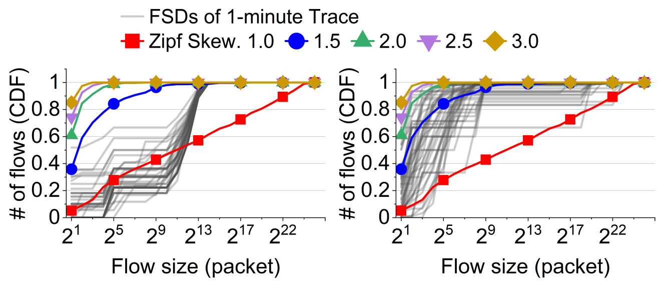
❷ FSDs of attack and benign traffic are different. To explore that, we analyze the FSDs of two benign datasets (CAIDA [2]) and nine malicious datasets (MACCDC [1], UNSW [32], and CIC [55], [56]) by comparing them with datasets generated by varying the Zipf skewness parameter from 0 to 5 [3]. Particularly, we use the Kolmogorov–Smirnov (KS) test to calculate the similarity of realistic and self-generated datasets and show the p-values in Fig. 3. A higher p-value means the dataset is closer to a certain skewness. As shown, the CAIDA (benign) traffic skewness is stably maintained between 1.5 and 2.5, whereas the skewness of attack traffic varies from 0.8 to 3.2. Therefore, the flow measurement accuracy of the cascaded multi-sketch approaches may not be robust upon a sudden change in the traffic patterns (i.e., under attack).

❸ We observe that FSDs of attack traffic can be completely different depending on how we define a flow (e.g., 1-, 2-, or 5-tuple), as shown in Fig. 3. For example, MACCDC's skewness varies from 1 to 2.3, and CIC 2019's skewness varies from 1.5 to 3 with different flow definitions.

Based on the above observations, we argue that even if a sketch is tailored to the skewness of a Zipf distribution (usually to the skewness of benign traffic), it is vulnerable to attack traffic with various distributions (Zipf with different parameters). One may actively configure the sketch to adapt to a certain attack FSD. However, the traffic distribution, especially attack traffic, is extremely unpredictable. More importantly, the sketch reconfiguration is extremely challenging for the in-network switch's data plane even with programmability.

B. In-network Switch and Data Plane Challenges

In-network programmable switches have emerged as a promising direction in defending against large-scale network attacks [4], [5], [31], [35], [39], [65]. Modern programmable switches fulfill Internet Service Providers' expectations for high-speed packet processing capacities (e.g., 25.6 Tbps [6]), high flexibility (e.g., customized function in the data plane), and low cost in comparison to the legacy switches [39]. The architectural design of the switch ASIC follows the Protocol Independent Switch Architecture (PISA) [9], which consists of a programmable parser and customizable match-action tables (i.e., network function table). Besides, SRAM register memory access (e.g., read/write) is supported via arithmetic logic units (ALUs) laid on the different stages of the ASIC hardware pipeline, allowing them to meet the functional requirements of a simple sketch data structure (e.g., Count-Min).



(a) UNSW DDoS HTTP trace (b) CIC 2019 trace

Fig. 2: FSD changes over time for the same stream. 49 FSDs of attack traces with one-minute window.

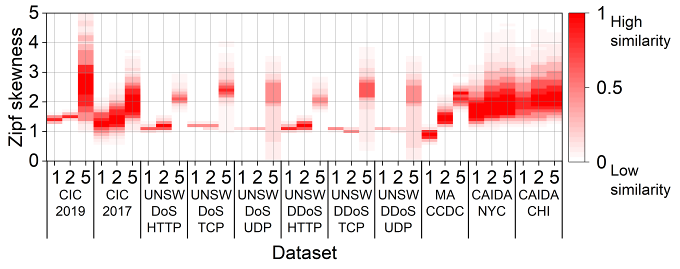


Fig. 3: FSDs of benign (CAIDA) and attack traces (MACCDC, UNSW, and CIC) compared with Zipf skewness from 0 to 5. The similarity (p-value) of Kolmogorov–Smirnov test is shown. A higher p-value means similar skewness. The traces are varied by flow definitions (1-, 2-, and 5-tuple).

To adapt to the potential change of FSDs, a sketch can reconfigure its data structure either online (based on a real-time measured FSD) or offline (based on the FSD of a measurement time window, i.e., epoch), both of which are challenging for in-network programmable switches. To date, several efforts have been made to dynamically reconfigure sketch data structure online (e.g., real-time merge of counters) [7], [18], [24]. Nevertheless, these solutions' algorithms were designed to work with general-purpose processors (i.e., CPU environments). Unlike in the general-purpose processors, however, a double-access to the same register, which is often required by the dynamic solutions, is not possible due to the pipeline design of the switch data plane [8], [31], [65]. Moreover, the high complexity of the online data structure configuration unavoidably adds overhead to the packet processing pipeline, which is unacceptable for high-speed switches. The offline approach reconfigures a sketch's data structure based on the FSD from the previous epoch. For instance, FCM can update the shape of its pyramid data structure to adapt to different FSDs. We note that such operations must be followed by memory re-allocations of the data structure in the data plane, requiring the entire data plane program to be recompiled and reloaded (i.e., a time-consuming process, say a few seconds). Therefore, the offline approach cannot adapt to FSDs changes in a timely manner.

C. Our Approach in a Nutshell

The eventual goal of Count-Less is to be robust to a broad range of traffic FSDs without reconfiguring the data structure in a switch's data plane. To adapt to various FSDs with the fixed data structure, a novel encoding/decoding algorithm is proposed for Count-Less based on an advanced data structure, called the pyramid-shape multi-layer counter. A similar data structure has been used in the past to reflect the Zipf distribution of traffic [58], [62], [63], [66]. Previous solutions constrained flows in a single layer and cascaded the counter value to the next layer when the current layer's counter is overflowed. Unlike these algorithms, our unique contribution, called the *cross-layer update with minimum update* algorithm, allows flows to utilize all the available counters across all layers for better survivability. Therefore, whatever an FSD looks like, the survival rate of each flow increases substantially (see section IV). Moreover, the *minimum update* strategy guarantees counters in each layer are not to be updated redundantly by a single packet. We stress that a different combination of data structures and encoding algorithms results in much more favorable behaviors of a sketch. To overcome the data plane constraints, we propose an ASIC-friendly approximation (i.e., pipeline design) for our optimal solution to fit Count-Less into the switch's data plane (see sections III and VI) for details.

III. The Count-Less Sketch: Design Space

In the following, we introduce Count-Less with our observations and design choices.

A. Data Structure

Count-Less's data structure is similar to Pyramid [63] and FCM [58]. A key idea in our data structure is splitting counters into smaller pieces to reduce the mouse flows' collision rate.

As shown in Fig. 4(a), Count-Less's consists of d layers of counter arrays, where each layer differentiates both the counter and the array size. For the counter size, the highest layer of Count-Less uses 32-bit counters for a sufficient counting range. The counter size is halved for each layer while going down to the lowest layer (left in the figure). With a large number of small counters at the lowest layer, the hash collision probability of flows at each layer is reduced significantly. Other works [58], [63] used a factor r to let a lower layer array possess r times more counters than its upper layer. In this work, we use $r = 4$ and $d = 3$ to fix the data structure and apply a *novel flow encoding strategy* to embrace a variety of FSDs.

B. Update Strategy and Encoding Algorithm

To explore the design space, we first consider our data structure choice (i.e., pyramid-shape) with the Count-Min's update strategy. The strategy updates a flow's counters at all layers regardless of the counter values (hereafter CL-CM), as shown in Fig. 4(a) (see CL-CM analysis in Section V-D).

CL-CM is expected to be more accurate than the standard Count-Min for mouse flow (e.g., $\leq$255) estimations but has higher relative errors for elephant flows (e.g., $>$255). In terms of the data structure, the mouse flow collision rate can be reduced significantly since more counters are assigned at layer-1. Algorithm-wise, the all-layer update strategy allows mouse flows to be hosted by idle counters at all layers for having more chances to survive. Nevertheless, CL-CM will have poorer accuracy for elephant flows because the massive amount of mouse flows will flood counters in all layers regardless of the counter value, increasing the noise level of larger flows. Thus, the first challenge is *how to prevent mouse flows from flooding other flows in the cross-layer update mechanism?*

Algorithm 1: Encoding and Decoding

```
1  Inputs: Layer $d$, width of each layer $w_l$.
2  $val_{min}$ =INT_MAX;
3  for $l$=1 to $d$ do
4      $idx_l = hash_l(pkt)\ mod\ w_l$;
5      if $C_l[idx_l] \neq overflow$ then
6          if $C_l[idx_l] < val_{min}$ then
7              $val_{min} = ++C_l[idx_l]$;
8          end
9      end
10 end
11 Return $val_{min}$; /* Decoding on the fly*/
```

Minimum Update: Inspired by the conservative update strategy of Count-Min [15], which updates only the smallest among three counters in three layers for a flow, we apply the conservative update with the pyramid-shaped data structure (CL-CU hereafter, see the analysis in Section V-D). While we consider CL-CU to be *optimal* in terms of accuracy, it is infeasible for a switch's data plane because of its pipeline design. Particularly, the conservative update algorithm first locates the smallest counter by reading d counters of a flow from all layers (i.e., different memory region) and has to *revisit* one of the counters for the update operation. However, the data plane works as a pipeline, which means double-access to a memory region is not allowed after the memory processing stage [8], [31], [65]. Therefore, the next challenge becomes *how to update the minimal counter value across all layers without memory double-access for data plane feasibility?*

Algorithm 1 and Fig. 4 show the encoding and decoding processes of Count-Less and examples of the *minimum update*. As shown in Algorithm 1, CL-MU's encoding and decoding are one combined operation, thus an estimation of a flow can be obtained while encoding, which enables the switch to detect anomalies on-the-fly (line 11). For each packet of a flow, CL-MU updates one counter at each layer following the lowest-to-highest order (line 3). Moreover, instead of updating counters at all layers as in Count-Min, CL-MU maintains a variable for storing the minimal value (val_{min}) upon the iterated layer (lines 2 and 7). Then, the *minimum update* is performed, which updates the layer only if the counter ($C_l[idx_l]$) is smaller than val_{min} (lines 6–7). The counter update event at a layer will be skipped when the layer counter exceeds its counting capacity (line 5).

Fig. 4 depicts CL-MU's four encoding scenarios (update): cross-layer, overflow, minimum, and worst-case update.

(a) Cross-layer: This scenario is shown in Fig. 4(a). For each packet, CL-MU updates its counters sequentially from layer-1 to layer-3 (①→②→③). As shown, once the first layer is increased by "1", the temporal minimal value (val_{min}) becomes 16, according to Algorithm 1. Since the second layer counter is smaller than val_{min}, the counter is increased and val_{min} remains unchanged. This process repeats until the last layer. As such, CL-MU's encoding is different from the cascading approach that stops the layer updates upon a successful counter update.

(b) Counter overflow: This scenario is shown in Fig. 4(b). The figure on the left shows a scenario where the first layer counter is overflowed (i.e., $=2^8$-1). In this case, layer-1 counter is skipped and the counter updates continue for the rest of layers with CL-MU's cross-layer update logic. We note that a counter is not decodable if it is overflowed, which means all flows that use the counter have to be decoded from the next layers. The right figure shows a case that layer-1 and layer-2 are overflowed and the packet updates layer-3 counter only. This scenario infers that the current flow may share the layer-2 counter with a larger flow that uses a different layer-3 counter. To sum up, both cases illustrate that CL-MU allows a smaller flow to use counters in the upper layers for better survivability (i.e., to be counted with a lower noise).

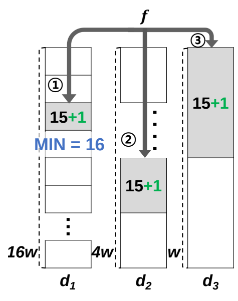
(c) Minimum update: This scenario is shown in Fig. 4(c). Intuitively, *minimum update* is an approximate version of the conservative update. The conservative update, however, cannot be implemented in an ASIC-based switch because the register cannot be read twice. To address the issue and get a similar result as the conservative update, *minimum update* maintains the current minimal value, not the global minimal value, while sequentially updating counters starting from layer-1. The update proceeds from the lower layer to the upper layer, because the lower layer has a smaller size but a larger number of counters. This means that a lower layer is likely to have a minimal value, and that the sequential update of the minimal value is likely to give us a value close to the global minimal.

As shown in Fig. 4(c)'s first part, per ①, layer-1 is not saturated, so it is updated, and the sequential minimal value becomes 201. In ②, the value of layer-2 is greater than the minimal value, so it is not updated. Upon that, layer-3 is checked, where the value is smaller than 201, thus incremented by 1. In the second figure, and upon updating layer-1 and layer-2, per ③, layer-3 has a larger value than the minimal value, and so it is not updated. This shows that unlike CM sketch (indefinite update) or FCM sketch (cascade update), *minimum update* prevents elephants from being contaminated by mice.

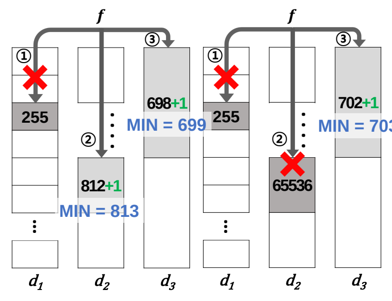
(d) Worst-case This scenario occurs when the counter of the upper layer has the smallest value, which is shown in Fig. 4(d). Per ①, the minimal value after layer-1 update is 106. Per ②, we update layer-2 and the minimal value will be 101, and layer-3 is updated as well, per ③. With the conservative update that updates with the global minimal value, only the value of layer-3 is increased. With *minimum update*, the values at layer-1 and layer-2 perform undesirable updates. However, due to the nature of Count-Less, there are fewer collisions and less error in the lower layer, since it has more counters and smaller counter size. Due to that, there is a high probability for having a smaller value in the lower layer, and the probability of such a worst-case update is small.

C. Advantages and Costs

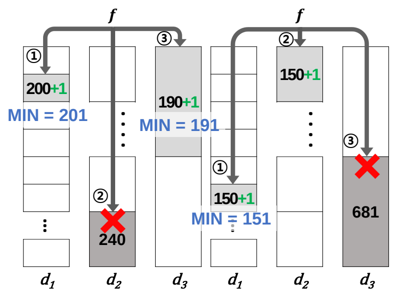
Unlike the multi-sketch cascaded approaches, which encode each flow at a single layer and cascade the overflowed counter value to the next layer, our cross-layer update strategy improves the flow survival rate by allowing all layers to be utilized for flows of any size. Meanwhile, by minimizing redundant counter updates across layers (i.e., minimum update), our approach achieves a tighter error bound compared to the Count-Min, which updates counters at all layers regardless of counter values. The flexible counter use in all layers allows CL-MU to accommodate a variety of FSDs.



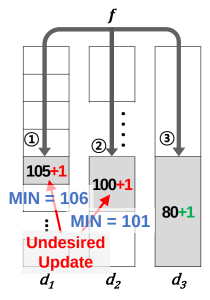
(a) Cross-layer update



(b) Overflow update



(c) Minimum update



(d) Worst case update

Fig. 4: Examples of update operations (①→②→③). (a) shows Count-Less (CL-MU) allows mouse flows (≤255) to survive at multiple layers with idle counters. (b) illustrates that saturated layers are not updated in the overflow case. (c) depicts that *minimum update* protects larger flows from mouse flow flooding. (d) demonstrates the worst case update scenario of *minimum update* due to the unawareness of the global minimal value.

The conservative update algorithm requires awareness of the global minimal value across all layers, which triggers the unacceptable double-access of the smallest counter for the pipeline-based switches. To address this issue, CL-MU approximates the global minimal value with the temporal minimal value upon the iterated layers. As shown in Fig. 4(d), this approximation sometimes results in undesired update at the initial layer (i.e., unawareness of the global minimum). However, we note that the approximation-caused error is negligible (see section V), whereas making the *minimum update* feasible in the data plane.

IV. Theoretical Analysis

In this section, we provide a theoretical analysis of Count-Less (CL-MU). In particular, the effects of the combination of pyramid data structure and *minimum update* are presented in terms of theoretical error bounds and probability. Since Count-Less follows the same update principle of the *cross-layer update*, we take advantage of Count-Min's theory to calculate the error and the probability. Table I shows the notations. From [10], Count-Min with an array of counters of width $w = e/\epsilon$ and depth $d = \ln 1/\delta$ has an error bound for a query f for each row as follows [10]; given parameters (ϵ, δ), where e is Euler's number.

Configuration of Count-Less sketch: Given the parameters (ϵ_d, δ), and unlike Count-Min sketch, Count-Less with *mini-*

TABLE I: Parameters used in Count-Less (CL-MU) sketch

Params	Description
n	the total number of flows
i	a flow ID
j	a layer of Count-Less
d	the number of layers
w_j	the width (i.e., number of counters) of a layer-j
cnt$[d][w_j]$	the counter array for Count-Less
δ	the probability of the ϵ_d-bounded error
ϵ_j	the error bound factor of the j-th layer
r	the ratio of width btw adjacent layers ($r = 4$)
T_j	the counter's limit of a layer-j ($T_j = 2^{j+1} - 1$)
a_i	the actual size of a flow i
$\hat{a}_i$	the estimated size of a flow i
$\vec{a}$	the vector of a_i's
a_i^j	the actual size of a flow i in the j-th layer
$\vec{a}^j$	the vector of a_i's in the j-th layer
$\hat{a}_i^j$	the estimated size of i recorded in the j-th layer

mum update has a variable number of counters for each layer; that is, $w_j = r^{d-j} w_d$ counters for layer $j = 1, 2, \ldots, d$, and $d = \ln 1/\delta$ layers for recording flow sizes. Here, $w_d = e/\epsilon_d$ is the number of counters in the last (d-th) layer, and r is Count-Less's expansion factor, for which *four* is recommended in this paper. Each counter in the j-th layer is 2^{j+1}-bit long. Because the counters in Count-Less's j-th layer has 2^{j+1} bits, the counting limit in each layer is $2^{j+1} - 1$. We denote the limits as $T_1, T_2, \ldots, T_{d-1}$ for the $d-1$ layers, respectively ($T_0 = 1$ for notational convenience). Because of the limits, the total number of the packets inserted (or encoded) varies by layer, and is denoted by:

$$\|\vec{a}^j\|_{1'} \overset{\text{def}}{=} \sum_{k=1, 1 \le a_k^j \le T_j}^{n} a_k^j, \tag{1}$$

We note that every L_1 norm of a in d layers are all the same for Count-Min. For cascaded sketches, such as *Cold Filter* and *Pyramid sketch*, only those flows with size that ranges from T_{j-1} to T_j are encoded, unlike Count-Less, which encodes flows from 1 to T_j by adopting *cross-layer update*. Because not every a_k^j is encoded additively and a mouse flow is always overwritten by an elephant flow by the *minimum update*, the actual value of $\|\vec{a}^j\|_{1'}$ is smaller than in the above definition.

In the following theoretical analysis, we focus on the probability to guarantee an estimation error bound. More specifically, targeting the estimation error bound of Count-Min sketch with the same number of layers and the same error bound factor, we prove what the probability is to guarantee the error bound in Count-Less, which is $\epsilon_d \|\vec{a}^d\|_{1'}$. Also, a discussion on how much probability gain can be obtained by Count-Less compared to Count-Min is followed.

Lemma 1. *Let $X_{i,j}$ be Count-Less's additive error of a flow i in the j-th layer, and ϵ_d is the error factor of the d-th layer. Then, the probability of the estimation to be bounded by ϵ_d fraction of the total packets encoded in the d-th layer is*

$$Pr[X_{i,j} > \epsilon_d \|\vec{a}^d\|_{1'}] < \frac{\epsilon_j}{e\epsilon_d}$$

Proof: Let a_i^j be the actual flow size of the flow i at layer j. Count-Less's estimation of a_i is $\hat{a}_i = \min_j(cnt[j, h_j(i)])$, where $h_j(i) \in [1, w_j = e/\epsilon_j], j \in [1, d = \ln 1/\delta]$. Because of the additive errors by other flows, $cnt[j, h_j(i)] = a_i^j + X_{i,j}$, where $X_{i,j}$ is a random variable representing the added error to a_i^j in the j-th layer.

To analyze the error term, we introduce an indicator variable $I_{i,k,j}$ to indicate if there is a hash collision in the index for two flows i and k ($k \neq i$) with the hash function h_j:

$$I_{i,k,j} = 1 \leftrightarrow h_j(i) = h_j(k) \text{ for } i \neq k \text{ and } I_{i,k,j} = 0 \text{ otherwise} \tag{2}$$

The indicator $I_{i,k,j}$ being a binary random variable, the expectation of $I_{i,k,j}$ is the probability that $I_{i,k,j} = 1$:

$$E(I_{i,k,j}) = \Pr[h_j(i){=}h_j(k)] = \frac{1}{\text{range}(h_j)} = \frac{1}{w_j} = \frac{\epsilon_j}{e}, \tag{3}$$

assuming that $h_j()$ is chosen from a family of universal hash functions. Using $I_{i,k,j}$, we can express the error term $X_{i,j}$ as:

$$X_{i,j} = \sum_{k=1}^{n} I_{i,k,j} * a_k^j = \sum_{k=1 | h_j(i)=h_j(k), 1 \le a_k^j \le T_j}^{n} a_k^j. \tag{4}$$

Because $I_{i,k,j}$ and a_k are independent, the expectation of the error is calculated as follows:

$$\begin{aligned} E(X_{i,j}) &= E(\sum_{k=1, 1 \le a_k^j \le T_j}^{n} I_{i,k,j} * a_k^j) \\ &= \sum_{k=1, 1 \le a_k^j \le T_j}^{n} a_k^j * E(I_{i,k,j}), \end{aligned} \tag{5}$$

Owing to equation (3), we can write the expectation as:

$$E(X_{i,j}) = \sum_{k=1, 1 \le a_k^j \le T_j}^{n} a_k^j * E(I_{i,k,j}) = \|\vec{a}^j\|_{1'} * \epsilon_j / e \tag{6}$$

By the Markov inequality and equation (6), the probability that the estimation error at a layer j is greater than $\epsilon_j \|\vec{a}^j\|_{1'}$ is then given as follows:

$$\begin{aligned} \Pr[X_{i,j} > \epsilon_d \|\vec{a}^d\|_{1'}] &\le \Pr[X_{i,j} \ge \epsilon_d \|\vec{a}^d\|_{1'}] \\ &\le \frac{\epsilon_j \|\vec{a}^j\|_{1'}}{e\epsilon_d \|\vec{a}^d\|_{1'}} < \frac{\epsilon_j \|\vec{a}^d\|_{1'}}{e\epsilon_d \|\vec{a}^d\|_{1'}} = \frac{\epsilon_j}{e\epsilon_d} \end{aligned} \tag{7}$$

■

Lemma 1 considers only the noise from a single layer independently, but Count-Less consists of d layers, which requires analysis of the probability considering all the layers for decoding error. Other than that, the probabilities are all different from each layer. Following theorems show the whole probability of the target error bound. To do so, we separate cases into two such as flow groups of less than or equal to T_{d-1} and of greater than T_{d-1}, and show their probability respectively.

Theorem 1 (Effect of Limited Cross Layer Update for Small flows)**.** *For a flow i of which the size ranges from T_{j-1} to T_j for $1 \le j \le d-1$, Count-Less estimates a_i by the following error bound with at least the following probability:*

$$Pr[\exists j, X_{i,j} < \epsilon_d \|\vec{a}^d\|_{1'}] > 1 - \frac{\gamma\delta}{e^{1-j} r^{d^c}}$$

where c is a constant of $1 \le c \le 2$, and $\gamma (\le 1)$ is the fraction of packets of the flows whose size range from T_{j-1} to T_j among the entire packets.

Proof: In Count-Less, the number of layers used by a flow varies depending on its size according to the cross-layer update strategy. Among d layers, a flow of which the size ranges from 1 to T_1 uses all the d layers, while one from T_1 to T_2 uses $d-1$ layers because the lowest layer is saturated quickly and does not update it anymore. Similarly, a flow of T_{d-1} or above uses only the highest layer of Count-Less. The probability that the error term is bounded is thus all different according to a flow size. We can divide the probability and the error size estimation into d cases with the corresponding thresholds $(T_1, T_2, \ldots, T_{d-1})$. However, we will show only two cases: the first case (i.e., the smallest flow group ($< T_1$), the j-th smallest flow group (between T_{j-1} and T_j), and the mid-sized flow group ($\geq T_{d-1}$), without any loss of generality.

Case 1) We consider a flow i in the smallest group that is less than T_1. In this case, a flow i does not saturate even the lowest layer and has d layers to record its size. By the pairwise independent hash assumption and Lemma 1, the probability that for all layers the error is greater than $\epsilon_d\|\vec{a}^d\|_{1'}$ is:

$$\Pr[\forall j, X_{i,j} > \epsilon_d\|\vec{a}^d\|_{1'}] < \prod_{j=1}^{d} \frac{E(X_{i,j})}{\epsilon_d\|\vec{a}^d\|_{1'}} = \frac{1}{e^d}\prod_{j=1}^{d} \frac{\epsilon_j\|\vec{a}^j\|_{1'}}{\epsilon_d\|\vec{a}^d\|_{1'}}. \tag{8}$$

Since $\epsilon_j = e/w_j$, Count-Less's pyramid shaped data structure ensure $\epsilon_j = \epsilon_d/r^{d-j}$. Moreover, $\|\vec{a}^j\|_{1'} << \|\vec{a}^d\|_{1'}$, because the elephant flows cannot reside in the j-th layer and the mouse flows only can if not collided with larger flows. Therefore,

$$\Pr[\forall j, X_{i,j} > \epsilon_d\|\vec{a}^d\|_{1'}] < \frac{1}{e^d}\prod_{j=1}^{d} \frac{1}{r^{d-j}}\frac{\|\vec{a}^j\|_{1'}}{\|\vec{a}^d\|_{1'}} < \frac{\delta}{r^{d^2}}. \tag{9}$$

The overestimation probability is then bounded by $\gamma\delta/r^{d^2}$ with the skewed data structure of Count-Less and the cross-layer update. Considering that Count-Less estimates a_i by the minimum among the counters as Count-Min does, the error of the j-th layer is bounded as follows:

$$\hat{a_i} < a_i + \epsilon_d\|\vec{a}^d\|_{1'} \tag{10}$$

with probability at least:

$$\Pr[\exists j, X_{i,j} < \epsilon_d\|\vec{a}^d\|_{1'}] > 1 - \frac{\delta}{r^{d^2}}, \tag{11}$$

Case 2) For size ranging from T_{j-1} to T_j, a flow i in this group already saturates the lower $j-1$ layers, and it is allowed to use $d-j+1$ layers. Due to Count-Less's minimum update policy, the flow group overrides any flow in the smaller flow groups, while larger flows saturate and do not use this layer anymore. From the view of a flow in this group, this case is equivalent to having the layer for its exclusive use, while sharing counters with smaller flows that are not collided. Thus, for a flow i, the number of packets of interest in this layer is $\|\vec{a}^j\|_{1'}$.

The j-th and $j+1, \ldots, d$-th layers are used for recording the flow size of this group, thus the probability that the error in all $d-j+1$ layers goes beyond the bound of Lemma 1 is:

$$\Pr[\forall s \in [j, \ldots, d], X_{i,s} > \epsilon_d\|\vec{a}^d\|_{1'_s}] < \prod_{s=j}^{d} \frac{E(X_{i,s})}{\epsilon_d\|\vec{a}^d\|_{1'}}. \tag{12}$$

Similarly,

$$\Pr[\forall j, X_{i,j} > \epsilon_d\|\vec{a}^d\|_{1'}] < \frac{1}{e^{d-j+1}}\prod_{s=j}^{d} \frac{1}{r^{d-j}}\frac{\|\vec{a}^j\|_{1'}}{\|\vec{a}^d\|_{1'}} < \frac{\gamma\delta}{e^{1-j}r^{d^c}}, \tag{13}$$

where c is a constant of $1 < c < 2$, and γ is the fraction of packets of the flows of which size range from T_{j-1} to T_j among the entire packets. Count-Less, therefore, estimates a_i as follows:

$$\hat{a_i} < a_i + \epsilon_d\|\vec{a}^d\|_{1'}, \tag{14}$$

with probability of at least

$$\Pr[\exists j, X_{i,j} < \epsilon_d\|\vec{a}^d\|_{1'}] > 1 - \frac{\gamma\delta}{e^{1-j}r^{d^c}}. \tag{15}$$

■

The recommended parameters for Count-Less are $r = 4$ and $d = 3$, which makes the probability much higher than Count-Min's $1-\delta$.

Theorem 2 (Effect of Minimum Update for Large flows)**.** *For a flow i of which the size ranges from T_{d-1} to T_d, Count-Less estimates a_i by the following error bound with at least the following probability:*

$$Pr[\exists j, X_{i,d} < \epsilon_d\|\vec{a}^d\|_{1'}] > 1 - \frac{\gamma}{e}$$

where c is a constant of $1 \leq c \leq 2$, and $\gamma(\leq 1)$ is the fraction of packets of the flows whose size range from T_{d-1} to T_d among the entire packets.

Proof: For a flow of size ranging from T_{d-1} to T_d, a flow i is in the largest group, which saturates every layer except the last, leaving one layer to record its size. Due to Count-Less's minimum update policy, the flow group overrides any flow in smaller flow groups. From the viewpoint of a flow in this group, this case is equivalent to having this layer for its exclusive use. However, unlike the multi-stage sketches, smaller flows without hash collision with the large flows still can record their sizes in this layer.

As such, to calculate the probability of the estimation error, we do not need to consider all the flows that reside in the highest layer (thanks to the minimum update strategy). It is, however, sufficient to consider only the elephant flows of which size is greater than T_{d-1}, resulting in:

$$\Pr[X_{i,d} > \epsilon_d\|\vec{a}^d\|_{1'}] < \frac{E(X_{i,d})}{\epsilon_d\|\vec{a}^d\|_{1'}} = \frac{\epsilon_d\gamma}{e\epsilon_d} = \frac{\gamma}{e}, \tag{16}$$

where γ is the fraction of packets of the elephant flows among the entire packets, which is small. Thus, Count-Less estimates a_i as:

$$\hat{a_i} < a_i + \epsilon_d\|\vec{a}^d\|_{1'}, \tag{17}$$

with a probability of at least:

$$\Pr[\exists j, X_{i,d} < \epsilon_d\|\vec{a}^d\|_{1'}] > 1 - \frac{\gamma}{e} \tag{18}$$

■

Considering that w_d is set as large as Count-Min's w, and the highest layer's counter size is 32-bits, γ/e is smaller than δ and the number of counters that can be exclusively taken is large. Consequently, flows in this group have hardly any collision in the layer, as will be shown in section V-D.

V. Evaluating Count-Less

In this section, we first analyze Count-Less's (CL's) robustness under various FSDs without adjusting data structure dynamically. Then, we provide a deep analysis of our novel encoding algorithm "*minimum update*" (CL-MU). Finally, we provide the evaluation of CL-MU with six security applications. All analyses are presented in contrast with various solutions, including (1) standard (Count-Min or CM), (2) naïve (CL-CM having the pyramid-shaped data structure with Count-Min update), (3) optimal (CL-CU having the pyramid shaped data structure with conservative update), and (4) state-of-the-art sketches (FCM [58] or Elastic [62]).

A. Parameters and Dataset

To show the memory impact, we vary the memory from 0.2 MB to 1 MB for all schemes. For CL-MU, we use a fixed parameter set (i.e., $d = 3$ and $r = 4$. Refer to Appendix E for the evaluation results with various parameters.) For fairness, we set the same number of layers ($d = 3$) for all multi-layer-based schemes, including all CM-based schemes, CL-MU, and FCM. When comparing CL-MU with FCM, we use the same layer expansion factor ($r = 4$ in CL-MU and $k = 4$ in FCM) for data structure (memory) configurations. Also, FCM uses two trees following the original work's setting [58]. For Elastic, we assign 150 KB of memory for its heavy part (i.e., hash table or TCAM) and the remaining memory is assigned for the light part (i.e., CM sketch with $d = 1$ in SRAM), as suggested in the original work [62]. We note that CL-MU and FCM do not require the support of TCAM and can work with SRAM only in a switch setting. In our experiments, we use three different datasets, which are as follows:

❶ To explore the robustness of CL-MU, we create network traces by varying the Zipf skewness from 1.0 to 3.4 [3]. Each trace contains 30 M packets. The traces are used to measure flow estimation accuracy of CL-MU under various skewnesses without data structure reconfiguration (see section V-C).

❷ To show the characteristics of CL-MU, we conduct in-depth analysis of CL-MU with one-minute real-world trace (i.e., CAIDA [2]) including 31.27M packets with 1.88M flows (see section V-D). In this work, we define the mouse flows as a flow with ≤ 255 packets and the elephant flows as those with >255 packets. As a result, the one-minute trace contains 99.2% of the mouse flows and 0.8% of the elephant flows.

❸ Finally, we use CL-MU to perform security measurement tasks with three datasets, the CAIDA dataset and two combined datasets that mixes the benign CAIDA trace (skewness≈2.0) with mouse-heavy (skewness≈3.0) and elephant-heavy (skewness≈1.0) attack traces [32], [56], respectively. For the benign trace, we use 32 continuous five-second CAIDA sub-traces by considering each sub-trace to be an epoch. Each sub-trace contains 2.5M to 2.7M packets with 226K to 244K distinct 5-tuple flows. On average, 99.15% of the flows are mice and 0.85% of them are elephants. To add a major impact to the benign trace, the attack traces match the total number of packets with the benign trace (see section V-E).

B. Evaluation Metrics

We use six metrics to evaluate CL-MU: Average Relative Error (ARE), Flow Survival Rate (FSR), Relative Error (RE), Weighted Mean Relative Error (WMRE), F1 Score, and Throughput, which we define in the following.

ARE: this is the averaged relative error, $\frac{1}{n}\sum_{i=1}^{n}|f_i - \hat{f}_i|/f_i$, where n is the number of flows and f_i and $\hat{f}_i$ are the actual and estimated flow sizes, respectively. ARE is used to evaluate the accuracy of the flow size estimation.

FSR: this metric is defined as the fraction of flows that are below a certain relative error after decoding. Considering the different noise impacts for different sized flows, we break the flow sizes into two ranges, mouse (1∼254) and elephant (255∼), and consider a flow as survived if the estimated relative error is below 0.1 and 0.01, respectively.

RE: this metric is defined as $\left|1 - \frac{estimated}{actual}\right|$, with the *actual* and *estimated* values, respectively. We use RE to evaluate the accuracy of the flow size, cardinality, and entropy estimations.

WMRE: this metric is defined as $\frac{\sum_{i=1}^{z}|n_i - \hat{n_i}|}{\sum_{i=1}^{z}(\frac{n_i+\hat{n_i}}{2})}$, where z is the maximum flow size, n_i and $\hat{n_i}$ are the actual and estimated numbers of flow size i, respectively [33]. We use WMRE to evaluate the accuracy of the flow size distribution.

F1 Score: this metric is defined as $2 \times \frac{precision \times recall}{precision + recall}$, where the *precision* is the ratio of the true instances among those reported and the *recall* is the ratio of the reported true instances. F1 score is used to evaluate the accuracy of the heavy hitter and heavy changer.

Throughput: this metric indicates the packet processing capacity in million packets per second (Mpps) (see appendix B).

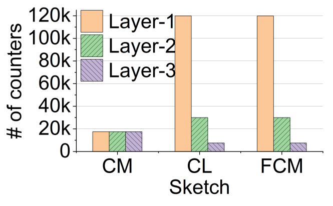
C. Robustness of Count-Less

We start evaluation with flow measurement robustness. Robust in-network measurement functions must have stable performance under various FSDs even in extreme cases (i.e., volumetric attacks). Therefore, we create dataset varying Zipf skewness (see section V-A ❶)) to examine the accuracy (i.e., ARE) of CL-MU sketch with a small memory space (i.e., 0.2 MB). For comparison, we use two state-of-the-art schemes, FCM [58] and Elastic [62] sketches, where the former varies its data structure parameter from $k = 4$ to $k = 32$ and the latter assigns 150 KB memory space for its (TCAM-assisted) Top-K filter.

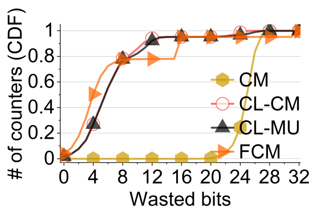
Recall that CL-MU expects to use *cross-layer update* strategy to enable flexible resource usage and *minimum update* algorithm to reduce all counters' noise level, and eventually provides better accuracy under various FSDs without data structure reconfiguration. Table II shows the accuracy of three schemes varying Zipf skewness. As shown, CL-MU shows the best accuracy for most cases except for the skewness 1.0 and 1.2. We note that the smaller skewness means more elephant flows exist in the network trace, thus the TCAM-based (i.e., exact counting) elephant filter of Elastic sketch unsurprisingly shows the best accuracy among three sketches. However, as the skewness increases, CL-MU outperforms all other schemes and consistently provides the best accuracy for skewness 1.4∼3.4. For FCM, a larger parameter k means more friendly its data structure to mouse flows (i.e., higher Zipf skewness). However, FCM's accuracy decreases significantly for higher k and skewness (i.e., 2.6∼3.4), as shown in Table II. This is

TABLE II: ARE according to the trace's Zipf skewness. Boldface represents CL-MU, and the shaded ones are the best.

Skewness	1.0	1.2	1.4	1.6	1.8	2.0	2.2	2.4	2.6	2.8	3.0	3.2	3.4
CL-MU	**0.01**	**1.48**	**3.52**	**15.14**	**47.97**	**102.38**	**151.41**	**184.69**	**203.40**	**216.39**	**223.00**	**230.40**	**231.07**
FCM ($k = 4$)	0.04	1.08	7.38	25.45	83.56	232.92	605.06	851.19	968.37	1041.11	1077.61	1116.88	1121.75
FCM ($k = 8$)	0.38	0.46	4.10	16.94	58.49	155.10	355.62	743.13	927.85	1017.87	1056.52	1102.57	1100.71
FCM ($k = 16$)	3.32	2.82	4.76	16.21	55.49	142.29	289.47	623.66	956.71	1145.30	1204.69	1275.16	1260.22
FCM ($k = 32$)	2.25	1.05	6.60	21.67	66.30	163.90	315.45	653.32	1080.33	1421.29	1552.29	1693.87	1661.38
Elastic	0.00	0.13	4.27	19.89	78.17	163.80	198.40	208.86	216.99	223.38	228.52	232.69	236.11

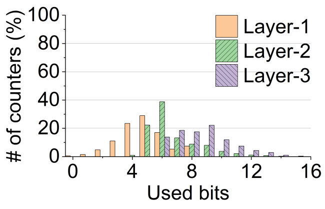


(a) Counter distribution

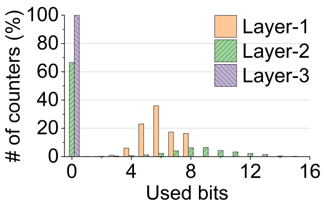


(b) Per-counter bit waste

Fig. 5: (a) and (b) compares CM, CL (CL-CM and CL-MU), and FCM in terms of the counter distribution and bit waste per counter, respectively. For fairness, CL and FCM have the same expansion factor ($r = 4$ in CL and $k = 4$ in FCM).



(a) CL-MU sketch



(b) FCM sketch

Fig. 6: Layer-wise comparison of CL-MU and FCM in terms of the bit-wise counter utilization.

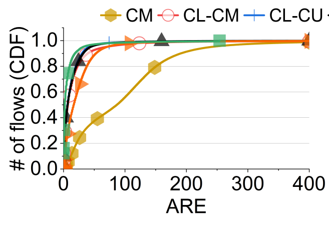
because its cascaded-sketch approach leads to a quick overflow of lower layer counters and all flows will be encoded/decoded at its last layer with a high collision rate. Therefore, FCM with k=4, which has relatively more counters at the last layer compared to FCM with larger k's, shows better accuracy. To these ends, Count-Less with *cross-layer update* and *minimum update* (i.e., CL-MU) shows the best robustness in terms of flow measurement accuracy, thus CL-MU is able to embrace dynamic traffic changes.

D. Analysis of Count-Less

We conduct an in-depth analysis to explain the robustness of Count-Less (CL-MU). The results are compared with the standard Count-Min (CM), our naïve approach (CL-CM), our optimal (CL-CU), and the state-of-the-art works (FCM [58] and Elastic [62]). In our analytical experiments of cross-layer and minimum updates, we fix the memory space to 0.6 MB (the median for 0.2∼1.0 MB with 0.2 increments) for each sketch to count a one-minute CAIDA trace (see §V-A ❷) for demonstrating CL-MU's characteristics. In our synergy effect experiment, we vary the memory space from 0.2 MB to 1.0 MB to show the memory impact. When comparing CL-MU with FCM, we conduct a layer-wise analysis to show the behaviors of their update strategies. Since Elastic uses one layer for its sketch (i.e., $d = 1$), we compare only its overall performance (i.e., ARE and FSR) with CL-MU and FCM sketches.

Fig. 5(a) shows the comparison of CM, CL (CL-CM and CL-MU), and FCM sketches in terms of the counter distribution with 0.2 MB memory (the distribution ratio for different memory sizes (0.4∼1.0 MB) is the same). As shown in Fig. 5(a), CM's data structure uniformly distributes the full-size (i.e., 32-bit) counters into three layers, whereas FCM and CL maintain counters in a pyramid-shape with smaller counters at lower layers (i.e., 16-bit counters at layer-2 and 8-bit counters at layer-1). The difference in the data structure results in FCM and CL having more counters than CM given the same memory space. Moreover, the decreased counter size at the lower layers can avoid the memory waste compared with CM, where most memory is reserved as the most significant bits (MSBs) of its full-size counters but never used. As can be seen in Fig. 5(b), about 89% of CM's counters are wasting more than 24 MSBs, which is equivalent to more than 75% of the entire memory. The pyramid-shaped data structure shows a much better memory utilization rate even though we still can observe the difference in memory usage among these approaches. We note that the difference in memory utilization is caused by their different flow encoding strategies. Compared with FCM, CL-MU and CL-CM achieve better memory efficiency due to the *cross-layer update* strategy. While CL-CM's memory usage is slightly higher than CL-MU, it is because CL-CM updates all layers' counters regardless of counter values leading to a high noise level. Unlike CL-CM, CL-MU applies a unique minimum update strategy to reduce the noise while performing the cross-layer update.

(1) Cross-layer update for collision relaxation: Fig. 6 compares the bit-wise counter usage in three layers of FCM and CL-MU. We collected the counter values in each layer and derived the bit-wise usage of counters (e.g., counter value 7 (0bx111) uses 3 bits of the counter). The high bit use rate can be employed to infer a hash collision rate, where a higher bit usage of counters indicates more collisions, since the flows share the counters. We note that due to the Zipf skewness of the CAIDA trace (i.e., skewness of ≈2.0), a small number of elephant flows and a large number of mouse flows exist in the dataset. As shown in Fig. 6(a), CL-MU consumes less than 8 bits in 83.9% of the counters at layer-2 and 49.9% of counters at layer-3. The result infers that CL-MU encodes mouse flows (i.e., $< 2^8 - 1$) not only at layer-1 but also at layer-2 and layer-3, thanks to the *cross-layer* update, which allows mouse flows to use idle counters at the upper layers to survive. In other words, the cross-layer update relaxes the hash collisions as a flow can be decoded from multiple layers. Therefore, CL-MU is robust even in an extreme case of mouse flow explosion (i.e., Zipf skewness>4.0). In contrast, the cascaded sketch-based FCM wastes 66.5% of counters at layer-2 and 99.9% of counters at layer-3, since it strictly partitions flow estimation in each layer based on the flow size. Therefore, FCM's flow collision rate becomes higher leading to a poor average relative error compared to CL-MU (see Fig. 7).


(a) Mouse flows (size $\leq$ 255)

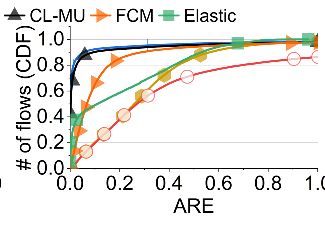
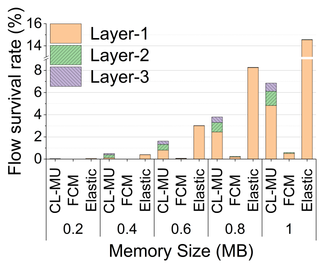

(b) Elephant flows (size $>$ 255)

Fig. 7: CDF of AREs of mouse and elephant flows.


(a) Mouse flow (size $\leq$ 255)

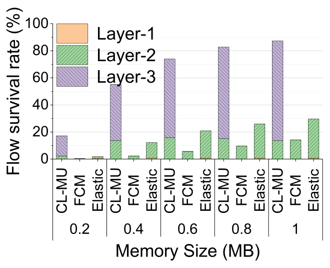

(b) Elephant flows (size $>$ 255)

Fig. 8: Flow survival rate comparison. A mouse flow is considered as survived if the estimated relative error is <0.1 and elephant flow is <0.01.

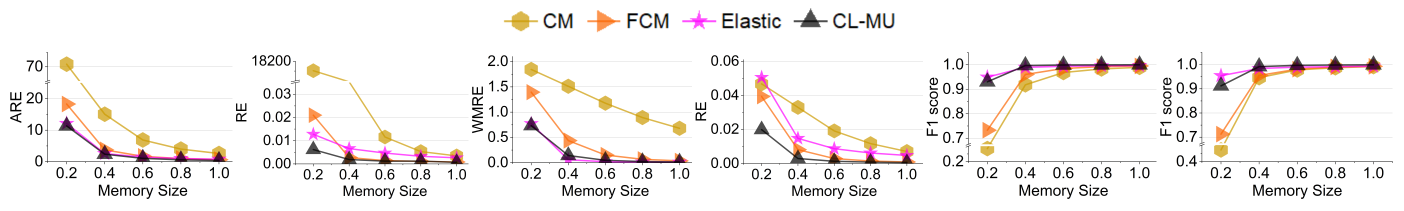
We note that the expected flow collision rate (or the expected number of collisions) can be given as $C \approx \frac{n_f(n_f-1)}{2M}$, where n_f is the number of distinct flows and M is the number of the possible encoding locations (i.e., counters) in the data structure. Since FCM encodes flows at a single layer at a time, the M_{FCM} is the number of counters in layer-1 in any case. On the contrary, CL-MU uses the cross-layer update strategy to encode flows using counters in all layers, which means $M_{\text{CL-MU}}$ is the total number of counters in all layers. Therefore, given the same amount of memory, $M_{\text{CL-MU}}$ is larger than M_{FCM}, thus FCM is expected to have more flow collisions compared to CL-MU (i.e., $C_{\text{FCM}} > C_{\text{CL-MU}}$). FCM has a higher bit-wise usage, since FCM uses two identical trees with different hash functions to encode all flows independently. Halved memory in each tree increases the hash collision probability, even with the same number of counters and hash function. The higher collision rate in the lower layer in spite of the plenty of counters, as shown in Fig. 6(b), comes from the Zipf distribution, where the number of elephant flows is much smaller than that of mouse flows.

For operational complexity (update time), CL-MU requires one hash calculation and three memory accesses per packet, whereas FCM needs two hash calculations and 2-6 memory accesses. As a result, FCM has one less memory access in the best case but one more hash calculation constantly.

(2) Minimum update for noise reduction: Subsequently, we explain the advantage of *minimum update* by comparing elephant flow noise level (i.e., ARE) among our CL-MU, CM, CL-CM, CL-CU, FCM, and Elastic sketch, as shown in Fig. 7(b). Per results, CL-CM and CM show the worst accuracy among the five schemes, since they both update all layers regardless of the counter values, which leads to the elephant flows' counters being flooded by all other flows that share the same counter at the same layer. As a result, the noise level of elephant counters is increased significantly. Next, FCM shows a better performance since it counts mouse flows only at the lowest layer and prevents them from flooding the upper layers (i.e., elephant flow counters). However, due to the cascading design, once a counter at the lower layer is overflowed, all the flows that share the counter will encode their packets at the upper layers. Due to the high bit-wise usage of counters at the lowest layer, as shown in Fig. 6(b), the counter overflow occurs frequently and leads to the mouse flows flooding the elephant counters. Then, Elastic performs best in mouse flow estimation since it uses a dedicated one-layer sketch with 8-bit counters to encode mouse flows. However, as shown in Fig. 7(b), Elastic's elephant flow estimation is the worst among the three advanced sketches due to the heavy load to be added to the memory scarce TCAM-based flow table. Finally, the proposed CL-MU achieves the same performance as the optimal CL-CU that is infeasible for pipeline-based switches. The result confirms that the minimum update strategy can significantly reduce the noise level for elephant flows by approximately updating the minimal counter of a flow among all layers. We note that such noise reduction effects are not only for the last layer only but also the lower layers by avoiding mouse flow flooding, as shown in our minimum update example in Fig. 4(c). In an extreme case of elephant flow explosion (i.e., Zipf skewness <1.0), the minimum update mechanism gives a higher priory to elephant flows at the last layer and prevents the mouse flooding as well. More importantly, mouse flows recorded in the last layer may be decoded from lower layers due to the *cross-layer* update.

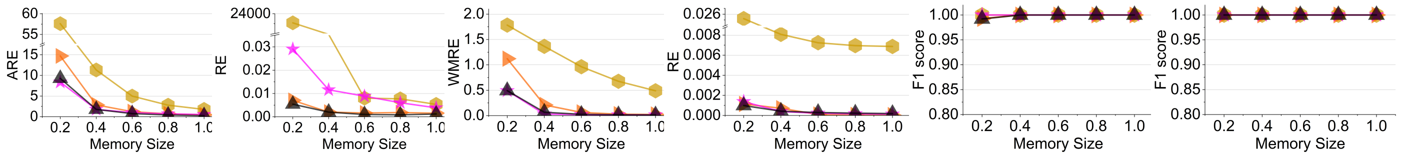
Synergy effect (robust flow measurement): To show the synergy effect of *cross-layer* and *minimum* update strategies, we introduce a new metric called flow survival rate (FSR). A flow is defined as "survived" if the flow estimation is lower than a certain RE. Also, we apply different error thresholds for different-sized flows (i.e., RE $<$ 0.1 for mouse flows ($\leq$255) and RE $<$ 0.01 for elephant flows ($>$255)). Eventually, FSR is defined as the number of survived flows over the total number of flows. More survived flows mean better measurement accuracy. We note that our definition of FSR is only theoretical to investigate CL-MU's properties, and is not meant for assessing security application. For instance, a 10-packet flow fails if the estimation is >11 (i.e., too harsh to be practical for security assessment). Fig. 8 shows the layer-wise FSR of CL-MU, FCM, and Elastic sketches. Since Elastic does not have three layers but two memory regions (i.e., SRAM for mouse flows and TCAM for elephant flows), we regarded SRAM (light part) as layer-1 and TCAM (heavy part) as layer-2.

It is worth noting that a flow in CL-MU can be decoded and survive at multiple layers, whereas a cascaded approach-based FCM sketch can decode flows only at one layer. Such multi-layer decoding option gives more chances to flows to survive, which is the key aspect to embrace for various FSDs (i.e., benign and attack traffic) without data structure reconfiguration. As shown in Fig. 8(a), CL-MU allows mouse flows to survive at all three layers using idle counters (i.e., FSR 0.03%∼6.85%), whereas FCM can decode mouse flows with a low survival rate (FSR 0.00%∼0.58%). Fig. 8(b) shows FSRs of elephant flows, and we observe that 17.15%∼87.24% of the elephant flows can survive in CL-MU at both layer-2 and layer-3, and only 0.38%∼14.21% survive in FCM at layer-2 and layer-3. Compared to CL-MU, as shown in Fig. 8(a), Elastic has roughly two times more mouse flows survived in layer-1,



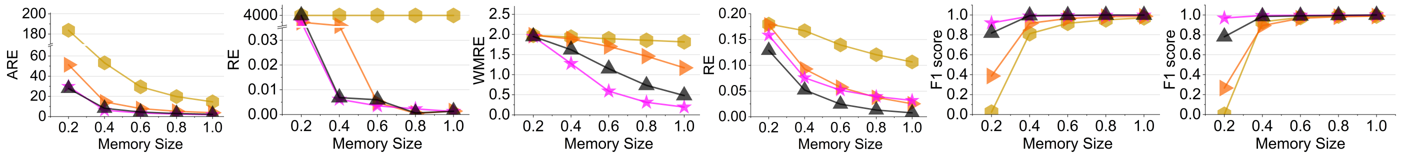
(a) Flow Size Estimation (b) Cardinality (c) Flow Size Distribution (d) Entropy (e) Heavy Hitter (f) Heavy Changer

Fig. 9: Security applications with benign (CAIDA) trace (Zipf skewness≈2.0) and compared among Count-Less (CL-MU), Count-Min (CM), Elastic, and FCM sketches. Memory usage from 0.2 MB to 1 MB.



(a) Flow Size Estimation (b) Cardinality (c) Flow Size Distribution (d) Entropy (e) Heavy Hitter (f) Heavy Changer

Fig. 10: Security applications with benign (CAIDA) and attack (UNSW) mixed trace (Zipf skewness≈1.0).



(a) Flow Size Estimation (b) Cardinality (c) Flow Size Distribution (d) Entropy (e) Heavy Hitter (f) Heavy Changer

Fig. 11: Security applications with benign (CAIDA) and attack (CIC) mixed trace (Zipf skewness≈3.0).

since it assigns a very small portion of memory for layer-2 (i.e., TCAM or elephant flow table) and most memory is allocated with 8-bit counters as a sketch (i.e., SRAM or mouse flow sketch). As a cost, Elastic sketch's FSR for elephant flows is much lower than CL-MU, as shown in Fig. 8(b). With the results, we can conclude that CL-MU benefits from both *Cross-layer* and *Minimum* update, allowing CL-MU to flexibly utilize the limited resources to count flows while providing a well-balanced accuracy between mouse and elephant estimation.

E. Security Applications

We show Count-Less's (CL-MU's) performance with network security measurement tasks, including flow size estimation, cardinality, flow size distribution, entropy, heavy hitter, and heavy changer. To show the robustness of CL-MU, we use three datasets in the experiments, namely *(i)* benign trace, *(ii)* benign trace with mouse-heavy attack, and *(iii)* benign trace with elephant-heavy attack (see section V-A ❸). Fig. 9∼Fig. 11 shows comparisons of CL-MU, Count-Min (CM), FCM [58], and Elastic sketch [62] with different metrics varying memory space from 0.2 MB to 1 MB.

Flow size estimation: Flow size estimation entails a per-flow packet counting of all individual flows in an epoch. As shown in Fig. 9(a), 10(a), and 11(a), CL-MU outperforms CM and FCM in all traces and memory settings. Also, it is more accurate than Elastic sketch in most settings, except for the elephant-heavy trace with 0.2 MB memory. The results are in line with our results in section V-C and V-D, and confirm stable performance of CL-MU with different scenarios.

Cardinality: In cardinality estimation, we count the number of distinct flows in a network trace. CL-MU uses linear counting [61] for the cardinality measurement (see appendix A for detail). As shown in Fig. 9(b), 10(b), and 11(b), CL-MU provides the most stable (i.e., more accurate or similar) cardinality estimation among four schemes with varying memory spaces and traces. As shown in Fig. 10(b), under the elephant-heavy attack, CL-MU's accuracy is slightly higher than FCM (i.e., 1.01∼2.19 times) and much more accurate than Elastic sketch (i.e., 3.26∼8.99 times). Also, CL-MU outperforms all other schemes in all memory settings in the mouse-heavy trace, as shown in Fig. 9(b). Lastly, FCM's RE with 0.4 MB memory is much poorer than CL-MU and Elastic sketches.

Flow size distribution: For the flow size distribution, CL-MU takes advantage of Model Reference Adaptive Control (MRAC) [33] algorithm (see appendix A for detail). As shown in Fig. 9(c), 10(c), and 11(c), CL-MU outperforms CM and FCM sketches always, achieving WMRE of 0.01∼0.73 for benign trace, 0.01∼0.49 for elephant-heavy trace, and 0.48∼1.94 for the mouse-heavy trace. Also, CL-MU and Elastic sketches' performance are comparable in the benign and elephant-heavy scenarios. Although Elastic sketch's WMRE is lower than CL-MU for the mouse-heavy trace, the result is precise enough to support further analysis (i.e., entropy, see appendix A). It is worth mentioning that Elastic and CL-MU sketches' WMREs drop significantly in the mouse-heavy scenario when increasing memory usage. This is because both sketches use small-size counter (i.e., 8-bit) for hosting mouse flows, thus the larger memory increases the number of counters significantly. This is favorable for Elastic sketch since it assigns more counters for mouse flows compared with CL-MU. Although FCM also uses small counters, its redundant tree structure suppresses the counter increment effects. Lastly, CM's counter increment effect is insignificant since it uses full-size counters (i.e., 32-bit) for its data structure.

Entropy: CL-MU uses MRAC also for the entropy estimation (see appendix A for detail). As shown in Fig. 9(d), 10(d), and 11(d), CL-MU shows better accuracy than all other schemes in benign and mouse-heavy scenarios. Particularly, CL-MU outperforms FCM by 2.05 times on average for the benign trace and 2.34 times for the mouse-heavy trace. We note that CL-MU can also utilize FSD as an input for the entropy estimation, which is faster than CL-MU with MRAC. One interesting observation is that even with FSD CL-MU is 1.33 times more accurate than FCM using MRAC. Also, CL-MU's performance is better than or similar to FCM and Elastic sketches.

Heavy hitter: For heavy hitter and changer detection, we assigned 0.1 MB of memory for a label table, which is responsible for storing large flows. In our experiments, a flow that contributes more than 0.01% (250$\sim$270) of the total number of packets in each epoch is defined as a heavy hitter and stored in our label table. As shown in Fig. 9(e), 10(e), and 11(e), Elastic sketch shows the best accuracy among four sketches, followed by CL-MU, FCM, and CM, respectively. We note that Elastic's high performance is supported by its exact counting-based elephant flow filter. However, since the elephant filter reserves a fixed amount of memory (150 KB), the amount of memory for mouse counters becomes too small given a memory space 0.2 MB, which degrades the accuracy of mouse flows, as shown in Table II.

Heavy changer: The heavy changers are flows whose size change (increases or decreases) is greater than a certain threshold over two adjacent epochs. Here, we use 0.01% of the total changes as a threshold. In the elephant-heavy scenario, all schemes show similar F1 scores since less mouse flows exist in the network trace, as shown in Fig. 10(f). In the benign and elephant-heavy traces, Elastic sketch achieves the best performance with 0.2 MB due to the same reason discussed in the heavy hitter evaluation. However, we observe that CL-MU achieves decent performance with more memory space (i.e., 0.4 MB$\sim$1.0 MB), as shown in Fig. 9(f) and 11(f).

Analysis: In the security applications, CL-MU has the most robust performance for flow size estimation, cardinality, flow size distribution, and entropy. FCM and Elastic sketches fail in either cardinality or entropy estimations for mouse-heavy or elephant-heavy attack scenario, with a single parameter setting, which convicts that they lack the robustness as network security function. In the heavy hitter and heavy changer detections, Elastic sketch provides the best accuracy since it pays more attention to the elephant flows rather than mouse flows (i.e., scarce TCAM-based exact counting). However, CL-MU's performance is comparable with Elastic sketch with more memory space. In FSR analysis as a theoretical and strict metric, we observe that CL-MU achieves a well-balanced flow estimation accuracy for both mouse and elephant flows, which is a key for robust security applications, each of which is sensitive to either mouse or elephant flow accuracy.

Part of the attack surface of sketches is the non-cryptographic hash function (i.e., CRC32) used in the randomization process during encoding. That is, an adversary may intentionally trigger hash collisions of flows in sketches. We note that while the pseudo hash function (CRC32) supported by switches is not as secure as the cryptographic hash function, its randomness is sufficient to support general network functions, including sketches. Moreover, the benefits from an intentional collision are limited from the perspective of detecting attacks by flow size [39], [65]. The intentional collision, possibly with a normal flow, makes the attack flow larger, which makes the attack flow (combined with a normal flow) easily detectable by the sketch-based traffic volume detectors.

VI. Data Plane Implementation and Evaluation

In this section, we present the implementation details and evaluations of our ASIC-friendly Count-Less (CL-MU). Particularly, we first describe the hardware implementation of CL-MU sketch in a programmable switch. Moreover, we compare CL-MU with the standard and state-of-the-art sketches in terms of resource usage, used stage, and packet processing latency. Lastly, we verify CL-MU's performance by reproducing the security application experiments using our prototype.

A. Implementations

We implemented CL-MU sketch in a Tofino Wedge-100 switch [14] in P4 language. For the data plane implementation, we added 115 lines of P4 code on a standard data plane program [48]. Meanwhile, the network traffic measurement applications are implemented in the switch's control plane using Bfrt Python [23]. Communication between data and control planes is realized through PCIe-based P4Runtime API [49]. The details of the data plane implementation are demonstrated and explained in appendix C.

B. Hardware Evaluation

In the following, we compare CL-MU with data plane-deployable solutions in terms of resource usage, stage, latency, and accuracy. We note that CL-MU P4 implementation was successfully compiled and executed in our switch, allowing CL-MU to process packets at line-rate [29], [42], [62].

Settings: In the hardware evaluation, we use 0.6 MB of memory for sketches. Our testbed consists of a Tofino programmable switch with 32x100 Gbps ports, and a network traffic generator that equips AMD Ryzen 5 2400 G 8-core CPU, 16 GB DRAM, and Intel 40 Gbps network adapter. libtins [17] library is used to replay CAIDA traces. Four security applications are executed in the control plane for analyzing sketch-collected data.

Resource usage: We discuss the resource usage of Count-Less (CL-MU), Count-Min (CM), CL-CM (Count-Less's data structure with CM's encoding algorithm) and FCM [58]. Elastic sketch's result is not included since it requires the scarce TCAM resource, and their P4 code is not publicly available. As shown in Table III, CM sketch consumes least resources among four schemes due to its simplicity. CL-MU requires slightly higher hash bits and SRAM compared to CM. CL-CM requires less ALU resources than CL-MU due to the simpler update algorithm. Lastly, FCM sketch's resource consumption is the highest. The high cost of FCM sketch is caused by its multi-tree design, which repeats the same process multiple times for independent sketches with different hash functions. Therefore, FCM's high ALU utilization indicates its complexity, which hurts the packet encoding latency of the

TABLE III: Comparison: data plane resource consumption, latency, and security application accuracy. Boldface represents Count-Less (CL-MU), and the shaded ones are the best.

Resource Usage	CM	CL-CM	CL-MU	FCM
Hash Bit (%)	2.88	3.06	**3.06**	4.97
SRAM (%)	5.72	6.14	**6.14**	7.29
ALU (%)	4.16	5.80	**6.25**	16.67
Used stages	2	4	**5**	4
Latency (Normalized)	CM	CL-CM	CL-MU	FCM
Total	0.09	0.75	**1.00**	1.75
Layer-1	0.02	0.09	**0.12**	0.09
Layer-2	0.02	0.42	**0.64**	0.75
Layer-3	0.05	0.24	**0.24**	0.91
Security Application	CM	CL-CM	CL-MU	FCM
Flow Size Estimation (ARE)	7.181	1.641	**1.171**	1.591
Cardinality (RE)	0.007	0.003	**0.002**	0.002
Flow Size Dist. (WMRE)	1.159	0.399	**0.041**	0.291
Entropy (RE)	0.018	0.009	**0.005**	0.006

data plane (see Table III). We note the SRAM usage difference is caused by the compiler, all schemes have the same memory size for their data structures.

Stage and packet processing latency: As shown in Table III, our advanced update algorithm sacrifices several stages for payment of our *minimum update* algorithm. CL-CM has the same stage numbers as FCM, but its latency is only 42.9% of FCM's, as shown in Table III. This result infers that the number of stages used in the data plane is irrelevant to the packet processing latency, such that CL-MU's latency is only 57.1% of FCM's with one more stage usage. As shown in Table III, CL-MU sketch's middle layer contributes the most to the total latency (63.41% on average) followed by the top layer's 24.39% and the bottom layer's 12.19%. However, the overall latency is much lower than that of FCM sketch, which contributes 52.37%, 42.05%, and 5.58% by layer-3, layer-2, and layer-1, respectively. This result suggests that appropriate separation of logic operations into more stages can reduce the processing latency, which is a better design than packing complex operations into fewer stages. Moreover, due to the resource management function of the data plane compiler, the idle resources at each stage can be optimized and assigned for other data plane functions to maximize the resource utilization.

Security applications in data plane: Table III shows accuracy comparisons among CM, CL-CM, CL-MU, and FCM sketches by varying security applications, namely flow size estimation (ARE), cardinality (RE), flow size distribution (WMRE), and entropy (RE). As shown, the standard CM performs the worst among the compared sketches. On the other hand, CL-MU provides the best overall performance. CL-CM outperforms CM and achieves similar accuracy with FCM, which supports our argument that the pyramid-shaped data structure is the major contribution factor of the accuracy improvement of cascaded multi-sketch approaches. Remarkably, with *minimum update*, CL-MU can improve the accuracy further based on the pyramid-shaped data structure.

VII. Related Work

To date, a large body of measurement systems are proposed, including OpenSketch [64], Dream [43], Pingmesh [22], UnivMon [38], Trumpet [44], FlowRadar [36], SketchVisor [25], Marple [45], Elastic sketch [62], and FCM [58]. Opensketch, FlowRadar, UnivMon, Elastic sketch, and FCM are generic and feasible for a programmable switch. Since introduced, Count-Min (CM) sketch and its variations [10], [20] play an important role in network traffic measurement. Especially thanks to its simplicity, CM can easily fit the constrained switch environment. Therefore, measurement systems such as Opensketch [64] and Elastic sketch [62] use CM as an essential component for security applications. However, one of the critiques about CM is the ignorance of the Zipf skewness in modern network traffic, which motivated a series of works [15], [28], [58], [62], [63], [66] that falls in a category of *multi-stage filtering*. Although these approaches achieve better accuracy than CM, the biggest contributing factor to their higher accuracy is their data structure design, which allocates smaller counters for mouse flows and fewer larger counters for elephant flows. While working for a certain flow size distribution, these solutions cannot work well under a variety of FDSs or sudden changes in traffic patterns (attacks) due to their fixed data structure. The other direction for the skewed stream is to manage data structure dynamically [7], [18], [24]. Unfortunately, these approaches are infeasible for a switch's ASIC due to their complexity and the switch's limited programmability. Mainly, Salsa [7] performs a left- or right-merge when a counter overflows. This merge operation requires revisiting the same register array when merging from 16-bit to 32-bit counter, violating the memory double-access constraint of pipeline-based switches.

Several directions have been explored to address the concern about the continuous use of sketches. One direction has been to recycle the sketches by periodically resetting them considering the transient counting values [27], [47], [53]. Another direction is to monitor flows over a short period of time using a sketch and fast memory (SRAM), and then to offload the counts into a large external memory (DRAM). For instance, LOFT [53] is a recent measurement framework that was designed specifically for programmable NICs (FPGA) working with CPU. The direct memory access (DMA) support in this environment allows the CM sketch to offload into external memory without delay. Since DMA is not supported by switches, the feasibility of FPGA-based solutions, such as LOFT, remains uncertain for switches. However, Count-Less and similar approaches can take advantage of the LOFT-like approaches to periodically offload measured flows into external memory for long-term and continuous measurement.

VIII. Conclusion

In-switch flow measurement still has a large room for improvement since the fixed data structure cannot adapt to sudden changes in traffic patterns (e.g., attacks). Moreover, the limited programmability of network switches obstructs dynamic approaches from bing deployed. To overcome these challenges, we propose a novel flow measurement scheme, namely Count-Less (CL-MU), to perform robust flow measurements in both benign and attack scenarios. Through extensive experiments and analyses, we show Count-Less can adapt to various FSDs and provides consistent performance for network security applications without re-configuring the data structure, thanks to our novel strategy (i.e., *minimum update*). Moreover, Count-Less is shown to be feasible for the switch's data plane with low computational and memory overheads.

Acknowledgment

The authors would like to thank the anonymous reviewers for their feedback. This work was supported by the Global Research Laboratory (GRL) Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (NRF-2016K1A1A2912757), by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (NRF-2020R1A2C2009372), and by Institute of Information and communications Technology Planning and Evaluation (IITP) grant funded by the Korea government(MSIT) (No.RS-2022-00155966, Artificial Intelligence Convergence Innovation Human Resources Development (Ewha Womans University)). DaeHun Nyang and Rhongho Jang are the corresponding authors.

References

[1] "Capture files from national cyberwatch mid-atlantic collegiate cyber defense competition (maccdc)," 2012. [Online]. Available: https://www.netresec.com/?page=MACCDC

[2] "The cooperative association for internet data analysis, equinix chicago data center," 2018, [13:00-14:00, Apr 19 2018., from Sao Paulo to New York]. [Online]. Available: https://www.caida.org

[3] "Fsd generation code with zipf distribution in stackoverflow," 2022. [Online]. Available: https://bit.ly/3vteXR1

[4] Y. Afek, A. Bremler-Barr, and L. Shafir, "Network anti-spoofing with SDN data plane," in *2017 IEEE Conference on Computer Communications, INFOCOM 2017, Atlanta, GA, USA, May 1-4, 2017*. IEEE, 2017, pp. 1–9.

[5] J. Bai, J. Bi, M. Zhang, and G. Li, "Filtering spoofed IP traffic using switching asics," in *Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos, SIGCOMM 2018, Budapest, Hungary, August 20-25, 2018*. ACM, 2018, pp. 51–53.

[6] I. Barefoot, *Intel® Tofino™ 3 Intelligent Fabric Processor*, Accessed August 8, 2022. [Online]. Available: https://www.intel.com/content/dam/www/central-libraries/us/en/documents/product-brief-final-version-pdf.pdf

[7] R. B. Basat, G. Einziger, M. Mitzenmacher, and S. Vargaftik, "Salsa: Self-adjusting lean streaming analytics," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 2021, pp. 864–875.

[8] R. Ben-Basat, X. Chen, G. Einziger, and O. Rottenstreich, "Efficient measurement on programmable switches using probabilistic recirculation," in *2018 IEEE 26th International Conference on Network Protocols, ICNP 2018, Cambridge, UK, September 25-27, 2018*. IEEE Computer Society, 2018, pp. 313–323.

[9] P. Bosshart, G. Gibb, H. Kim, G. Varghese, N. McKeown, M. Izzard, F. A. Mujica, and M. Horowitz, "Forwarding metamorphosis: fast programmable match-action processing in hardware for SDN," in *ACM SIGCOMM 2013 Conference, SIGCOMM 2013, Hong Kong, August 12-16, 2013*, D. M. Chiu, J. Wang, P. Barford, and S. Seshan, Eds. ACM, 2013, pp. 99–110.

[10] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.

[11] D. Dao, R. Jang, C. Jung, D. Mohaisen, and D. Nyang, "Minimizing noise in hyperloglog-based spread estimation of multiple flows," in *52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2022, Baltimore, MD, USA, June 27-30, 2022*. IEEE, 2022, pp. 331–342.

[12] F. Deng and D. Rafiei, "New estimation algorithms for streaming data: Count-min can do more," *Webdocs. Cs. Ualberta. Ca*, 2007.

[13] Y. Du, H. Huang, Y.-E. Sun, S. Chen, and G. Gao, "Self-adaptive sampling for network traffic measurement," in *IEEE INFOCOM 2021*, 2021.

[14] "Wedge 100bf-32x," Edgecore Networks, 2020. [Online]. Available: https://bit.ly/2YDeyv2

[15] C. Estan and G. Varghese, "New directions in traffic measurement and accounting," in *ACM SIGCOMM 2002*, 2002, pp. 323–336.

[16] P. Flajolet and G. N. Martin, "Probabilistic counting algorithms for data base applications," *Journal of computer and system sciences*, vol. 31, no. 2, pp. 182–209, 1985.

[17] M. Fontanini, "libtins: Packet crafting and sniffing library," http://libtins.github.io/, 2020.

[18] J. Gong, T. Yang, Y. Zhou, D. Yang, S. Chen, B. Cui, and X. Li, "Abc: a practicable sketch framework for non-uniform multisets," in *2017 IEEE International Conference on Big Data (Big Data)*. IEEE, 2017, pp. 2380–2389.

[19] M. Goswami, D. Medjedovic, E. Mekic, and P. Pandey, "Buffered count-min sketch on SSD: theory and experiments," vol. 112, pp. 41:1–41:15, 2018.

[20] A. Goyal, H. Daumé III, and G. Cormode, "Sketch algorithms for estimating point queries in nlp," in *In Joint Conference on EMNLP/CoNLL 2012*, 2012, pp. 1093–1103.

[21] A. Goyal and H. D. III, "Approximate scalable bounded space sketch for large data NLP," in *EMNLP 2011*. ACL, 2011, pp. 250–261.

[22] C. Guo, L. Yuan, D. Xiang, Y. Dang, R. Huang, D. A. Maltz, Z. Liu, V. Wang, B. Pang, H. Chen, Z. Lin, and V. Kurien, "Pingmesh: A large-scale system for data center network latency measurement and analysis," in *ACM SIGCOMM 2015*. ACM, 2015, pp. 139–152.

[23] F. Hauser, M. Häberle, D. Merling, S. Lindner, V. Gurevich, F. Zeiger, R. Frank, and M. Menth, "A survey on data plane programming with p4: Fundamentals, advances, and applied research," *arXiv preprint arXiv:2101.10632*, 2021.

[24] N. Hua, B. Lin, J. Xu, and H. Zhao, "Brick: A novel exact active statistics counter architecture," in *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, 2008, pp. 89–98.

[25] Q. Huang, X. Jin, P. P. C. Lee, R. Li, L. Tang, Y. Chen, and G. Zhang, "Sketchvisor: Robust network measurement for software packet processing," in *ACM SIGCOMM 2017*. ACM, 2017, pp. 113–126.

[26] "Intel streaming simd extensions (sse) technologies," Intel, 2020. [Online]. Available: https://software.intel.com/sites/landingpage/IntrinsicsGuide/

[27] R. Jang, D. Min, S. Moon, D. Mohaisen, and D. Nyang, "Sketchflow: Per-flow systematic sampling using sketch saturation event," in *IEEE INFOCOM 2020*. IEEE, 2020, pp. 1339–1348.

[28] R. Jang, S. Moon, Y. Noh, A. Mohaisen, and D. Nyang, "Instameasure: Instant per-flow detection using large in-dram working set of active flows," in *39th IEEE International Conference on Distributed Computing Systems, ICDCS 2019, Dallas, TX, USA, July 7-10, 2019*. IEEE, 2019, pp. 2047–2056.

[29] X. Jin, X. Li, H. Zhang, N. Foster, J. Lee, R. Soulé, C. Kim, and I. Stoica, "Netchain: Scale-free sub-rtt coordination," in *NSDI 2018*, 2018, pp. 35–49.

[30] X. Jin, X. Li, H. Zhang, R. Soulé, J. Lee, N. Foster, C. Kim, and I. Stoica, "Netcache: Balancing key-value stores with fast in-network caching," in *SOSP 2017*, 2017, pp. 121–136.

[31] C. Jung, S. Kim, R. Jang, D. Mohaisen, and D. Nyang, "A scalable and dynamic acl system for in-network defense," in *CCS '22: 2022 ACM SIGSAC Conference on Computer and Communications Security, Los Angeles, U.S.A., November 7-11, 2022*. ACM, 2022, pp. 1–13.

[32] N. Koroniotis, N. Moustafa, E. Sitnikova, and B. Turnbull, "Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset," *Future Generation Computer Systems*, vol. 100, pp. 779–796, 2019.

[33] A. Kumar, M. Sung, J. Xu, and J. Wang, "Data streaming algorithms for efficient and accurate estimation of flow size distribution," *ACM SIGMETRICS PER*, vol. 32, no. 1, pp. 177–188, 2004.

[34] A. Kumar and J. J. Xu, "Sketch guided sampling - using on-line estimates of flow size for adaptive data collection," in *INFOCOM 2006*. IEEE, 2006.

[35] Â. C. Lapolli, J. A. Marques, and L. P. Gaspary, "Offloading real-time ddos attack detection to programmable data planes," in *IFIP/IEEE International Symposium on Integrated Network Management, IM 2019,*

Washington, DC, USA, April 09-11, 2019, J. Betser, C. J. Fung, A. Clemm, J. François, and S. Ata, Eds. IFIP, 2019, pp. 19–27.

[36] Y. Li, R. Miao, C. Kim, and M. Yu, "Flowradar: A better netflow for data centers," in *NSDI 2016*. USENIX Association, 2016, pp. 311–324.

[37] Z. Liu, R. Ben-Basat, G. Einziger, Y. Kassner, V. Braverman, R. Friedman, and V. Sekar, "Nitrosketch: Robust and general sketch-based monitoring in software switches," in *ACM SIGCOMM 2019*, 2019, pp. 334–350.

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

[39] Z. Liu, H. Namkung, G. Nikolaidis, J. Lee, C. Kim, X. Jin, V. Braverman, M. Yu, and V. Sekar, "Jaqen: A high-performance switch-native approach for detecting and mitigating volumetric ddos attacks with programmable switches," in *USENIX Security 2021*, M. Bailey and R. Greenstadt, Eds., 2021, pp. 3829–3846.

[40] Y. Lu, A. Montanari, B. Prabhakar, S. Dharmapurikar, and A. Kabbani, "Counter braids: a novel counter architecture for per-flow measurement," in *Proc. ACM SIGMETRICS 2008*, Z. Liu, V. Misra, and P. J. Shenoy, Eds.

[41] A. Metwally, D. Agrawal, and A. El Abbadi, "Efficient computation of frequent and top-k elements in data streams," in *International conference on database theory*. Springer, 2005, pp. 398–412.

[42] R. Miao, H. Zeng, C. Kim, J. Lee, and M. Yu, "Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics," in *ACM SIGCOMM 2017*, 2017, pp. 15–28.

[43] M. Moshref, M. Yu, R. Govindan, and A. Vahdat, "DREAM: dynamic resource allocation for software-defined measurement," in *ACm SIGCOMM 2014*. ACM, 2014, pp. 419–430.

[44] ——, "Trumpet: Timely and precise triggers in data centers," in *ACM SIGCOMM 2016*. ACM, 2016, pp. 129–143.

[45] S. Narayana, A. Sivaraman, V. Nathan, P. Goyal, V. Arun, M. Alizadeh, V. Jeyakumar, and C. Kim, "Language-directed hardware design for network performance monitoring," in *ACM SIGCOMM 2017*. ACM, 2017, pp. 85–98.

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

[47] D. Nyang and D. Shin, "Recyclable counter with confinement for real-time per-flow measurement," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 3191–3203, 2016.

[48] "A standard p4 switch program," The P4 Language Consortium, 2020. [Online]. Available: https://github.com/p4lang/switch

[49] "The p4runtime api," P4 Language Consortium, 2021. [Online]. Available: https://p4.org/p4-runtime/

[50] W. W. Peterson, W. Peterson, E. Weldon, and E. Weldon, *Error-correcting codes*. MIT press, 1972.

[51] G. Pitel and G. Fouquier, "Count-min-log sketch: Approximately counting with approximate counters," *International Symposium on Web Algorithms*, 2015.

[52] G. Pitel, G. Fouquier, E. Marchand, and A. Mouhamadsultane, "Count-min tree sketch: Approximate counting for nlp," 2016. [Online]. Available: https://arxiv.org/abs/1604.05492

[53] S. Scherrer, C. Wu, Y. Chiang, B. Rothenberger, D. E. Asoni, A. Sateesan, J. Vliegen, N. Mentens, H. Hsiao, and A. Perrig, "Low-rate overuse flow tracer (LOFT): an efficient and scalable algorithm for detecting overuse flows," in *40th International Symposium on Reliable Distributed Systems, SRDS 2021, Chicago, IL, USA, September 20-23, 2021*. IEEE, 2021, pp. 265–276.

[54] "sFlow," http://www.sflow.org/.

[55] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization." in *ICISSp*, 2018, pp. 108–116.

[56] I. Sharafaldin, A. H. Lashkari, S. Hakak, and A. A. Ghorbani, "Developing realistic distributed denial of service (ddos) attack dataset and taxonomy," in *2019 International Carnahan Conference on Security Technology (ICCST)*. IEEE, 2019, pp. 1–8.

[57] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford, "Heavy-hitter detection entirely in the data plane," in *SOSR 2017*, 2017, pp. 164–176.

[58] C. H. Song, P. G. Kannan, B. K. H. Low, and M. C. Chan, "Fcm-sketch: Generic network measurements with data plane support," in *CoNext 2020*. ACM, 2020, pp. 78–92.

[59] R. H. M. Sr., "Counting large numbers of events in small registers," *Commun. ACM*, vol. 21, no. 10, pp. 840–842, 1978.

[60] M. Tirmazi, R. B. Basat, J. Gao, and M. Yu, "Cheetah: Accelerating database queries with switch pruning," in *ACM SIGMOD 2020*, D. Maier, R. Pottinger, A. Doan, W. Tan, A. Alawini, and H. Q. Ngo, Eds. ACM, 2020, pp. 2407–2422.

[61] K. Whang, B. T. V. Zanden, and H. M. Taylor, "A linear-time probabilistic counting algorithm for database applications," *ACM Trans. Database Syst.*, vol. 15, no. 2, pp. 208–229, 1990.

[62] 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 *ACM SIGCOMM 2018*, 2018, pp. 561–575.

[63] T. Yang, Y. Zhou, H. Jin, S. Chen, and X. Li, "Pyramid sketch: a sketch framework for frequency estimation of data streams," *Proc. VLDB Endow.*, vol. 10, no. 11, pp. 1442–1453, 2017.

[64] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with opensketch," in *NSDI 2013*, N. Feamster and J. C. Mogul, Eds. USENIX Association, 2013, pp. 29–42.

[65] M. Zhang, G. Li, S. Wang, C. Liu, A. Chen, H. Hu, G. Gu, Q. Li, M. Xu, and J. Wu, "Poseidon: Mitigating volumetric ddos attacks with programmable switches," in *NDSS 2020*. The Internet Society, 2020.

[66] Y. Zhou, T. Yang, J. Jiang, B. Cui, M. Yu, X. Li, and S. Uhlig, "Cold filter: A meta-framework for faster and more accurate stream processing," in *SIGMOD/PODS 2018*, 2018, pp. 741–756.

Appendix

A. Count-Less's Functions for Security Applications

We describe details of Count-Less (CL-MU) in estimating cardinality, flow size distribution, and entropy for security applications.

Cardinality: CL-MU uses its own data structure and the linear counting (LC) theory to estimate the cardinality [61]. To do so, we assume the lowest layer of CL-MU to be the LC's bit array. Then, the cardinality is calculated following the LC's equation, $\hat{n} = -s \cdot \ln(V)$, where s is the number of counters and V is the fraction of the empty (zero) counters in the array. Since CL-MU's number of counters at the lowest layer is 2x-16x larger than CM, CL-MU is saturated relatively slower. Given LC's performance is guaranteed only until the number of empty counters is less than 30%, CL-MU can estimate the cardinality with smaller memory.

Flow size distribution: To estimate flow size distribution, we measure the distribution of mouse flows and elephant flows separately. For mouse flows, we use the MRAC algorithm [33] with our layer-1 counters after setting the overflowed counters to "0". Subsequently, elephant flows will be involved by querying our sketch with the flow labels in the heavy hitter table.

Entropy: The entropy estimation can be done using one of two ways (1) using flow size distribution (Elastic sketch [62]) or (2) counter values (MRAC [33]). In section V-E, we showed the entropy estimation results of CL-MU with MRAC. With the flow size distribution, the entropy is calculated by leveraging the distribution array by $-\sum i * \frac{n_i}{N} * log\frac{n_i}{N}$, where N is the total number of flows and n_i is the number of flows with i packets [62]. In section V-E, we observed that Elastic's flow size distribution is more accurate than CL-MU's. Here, we

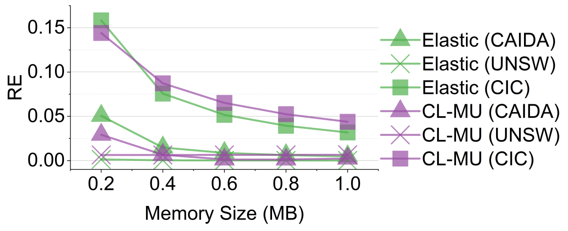


Fig. 12: Comparison: entropy estimation of CL-MU and Elastic with the flow size distribution.

show that CL-MU's entropy estimation based on the flow size distribution is comparable with Elastic, as shown in Fig. 12.

B. Throughput in CPU Environment

We measure sketch's throughput in a CPU environment to demonstrate the relative encoding complexity of different schemes. As shown in Fig. 13, CL-MU's throughput is roughly twice higher than the other sketches. CL-MU achieves the highest throughput of 18.13 Mpps. FCM sketch shows better throughput (16.42 Mpps) than CM and Elastic sketches. CL-MU and FCM are faster than CM since both require only one hashing by taking advantage of the pyramid-shaped data structure. The reason CL-MU is a bit faster than FCM is that FCM uses two trees and the counter updates occur twice at each layer. Elastic sketch posted the lowest throughput of 5.68 Mpps. However, we note that Elastic sketch was designed to estimate the source IP-based flows (i.e., 32-bit label). Therefore, the discrepancy between Elastic's throughput and the results posted in the original work is due to operating with the 5-tuple label (i.e., 104 bits), which cannot be accelerated by Intel's Streaming SIMD Extensions (SSE) technologies [26].

C. Implementation

In the following, we demonstrate the data plane implementation of CL-MU in detail. Our optimal CL-CU's update algorithm is composed of a two-step operation, namely (S1) reading counters of a flow in all layers and (S2) updating the smallest counter among them. In a switch's ASIC, the step (S2) requires revisiting the register array (i.e., layer) that hosts the smallest counter for updating. Unfortunately, such double-access of the same memory register is not allowed after the processing stage jumping to the next memory region (layer) during the step (S1) due to the pipeline design [8], [31], [65]. One can resolve the issue by leveraging packet re-circulation design, but it will hurt the bandwidth of the switch, which may not be acceptable for use in a high-speed environment [29], [30], [57].

Data plane: To overcome the challenge, we design the ASIC-friendly CL-MU, which is an approximate version of CL-CU, to fit the switch's pipeline. For data structure, each layer's counter array is implemented by register arrays and hosted by a single stage of the ASIC's packet processing pipeline. Moreover, the interaction operations between layers are integrated into the register action logic. Simply put, each layer was deployed sequentially in different match+action units (MAUs) over the pipeline. Fig. 14 illustrates the data plane logic of CL-MU. Three CRC32-based hash functions [50] are used to locate the counters for lookup and update of counter

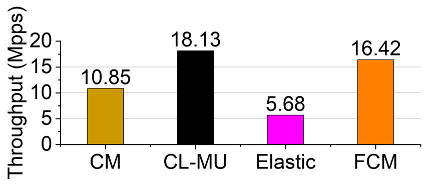


Fig. 13: Throughput: packet encoding capacity in a CPU environment. Mpps: million packets per second.

values (*CVs*) of a flow f in each layer. During the first stage of layer update, it checks if the current flow's counter reaches its counting capacity. If the counter is overflowed, we skip the counter update of the current stage (layer) and go to the next stage for the next layer's counter update. Otherwise, we increase the CV by "1". In the meantime, we read the pre-initialized minimal value (min) from the user metadata bus through the gateway and update min with CV. We note that all these actions are performed with the arithmetic logical units (ALUs) that are associated with the current stage's MAU. Next, the updated min will be stored back in the user metadata and passed to a subsequent stage (MAU). Similar operations are repeated at the next stage (i.e., layer-2) with an additional min update compared with the flow's layer-2 counter CV. At the last stage, we assume the counter cannot be overflowed due to a sufficiently large register (i.e., 32-bit). Therefore, we repeat the rest of the processes of layer-2. The P4 code snippet of CL-MU is shown in appendix D.

Control plane: The application's estimation functions reside in the switch's control plane. To retrieve statistics from the data plane, the control plane application fetches register values of each layer and heavy hitter table through P4Runtime API [49] for post-hoc analysis.

D. P4 Code Snippet

Our P4 code snippet of the essential logic of CL-MU includes the encoding/decoding functions of three layers, as shown in Fig. 15. Since our code is successfully compiled in to the switch's data plane, it guarantees line-rate packet processing [29], [42], [62].

E. Count-Less (CL-MU) with various parameters

In the following, we explain the rationale of our choice of our fixed parameters: the number of layers ($d = 3$) and the layer-wise expansion factor ($r = 4$). The former defines the number of layers (i.e., memory partitions) given a fixed memory space and the latter determines how many counters (memory) will be allocated in each layer (see section III for details of the parameters). To find out the best parameters, we vary d from 2 to 5 and r from 2 to 16, and analyze the accuracy of the mouse (≤255) and elephant (>255) flows separately. Fig. 16 illustrates the average relative error (ARE) of the mouse and elephant flows by varying memory space (i.e., 0.1 MB, 0.5 MB, and 1.0 MB) and two parameters (i.e., d and r).

Selection of r: As shown in Fig. 16(a), (c), and (e), regardless of the number of layers (d) and memory space, the layer-wise expansion factor $r = 4$ (red bars) shows the best overall performance for the mouse flows. Interestingly, although it is expected that a larger r ($r = 8$ depicted in blue bars and

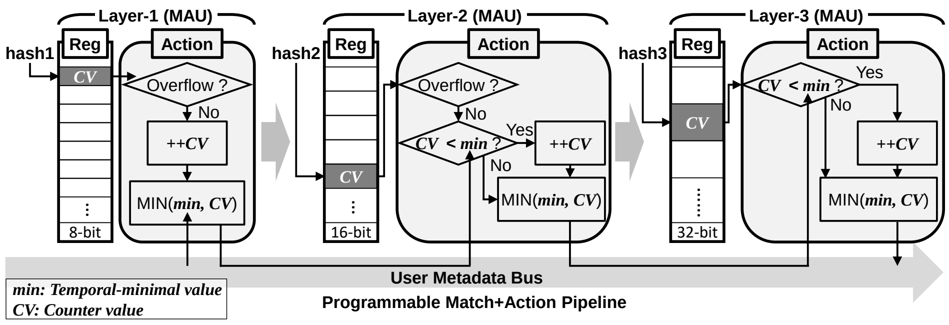


Fig. 14: Pipeline design of Count-Less: encoding/decoding logic of CL-MU in switch's data plane.

```
RegisterAction<8b, 32b, 8b>(cl1_register) cl1_action = {
  void apply (inout 8b value, out 8b result) {
    if (value>8w14) {value=8w15; result=meta.min}
    else {
      value = value + 8w1;
      result = min(meta.min[7:0], value);
  } } };

RegisterAction<16b, 32b, 16b>(cl2_register) cl2_action = {
  void apply (inout 16b value, out 16b result) {
    if (value>8w65534) {value=8w65535; result=meta.min}
    else {
      if (meta.min > value) {value = value + 16w1;}
      result = min(meta.min[15:0], value);
  } } };

RegisterAction<32b, 32b, 32b>(cl3_register) cl3_action = {
  void apply (inout 32b value, out 32b result) {
    if (meta.min > value) {value = value + 32w1;}
    result = min(meta.min, value);
  } };

apply {
  meta.min = (bit<32>)cl1_action.execute((bit<32>)hash1[17:0]);
  meta.min = (bit<32>)cl2_action.execute((bit<32>)hash2[25:10]);
  meta.min = (bit<32>)cl2_action.execute((bit<32>)hash3[31:17]);
}
```

Fig. 15: Data plane implementation of CL-MU in P4.

$r = 16$ with green bars) favors the mouse flows by allocating more small counters to the lower layers, the larger r helps relax the horizontal hash collisions only (among the same layer). Less counters (memory) allocated to the upper layer disallow the mouse flows to take advantage of Count-Less (CL-MU's) cross-layer update strategy that relaxes the collisions in a vertical manner (i.e., across layers), as shown in Fig. 16(a), (c), and (e). For the elephant flow estimation, a smaller r (i.e., $r = 2$ with gray bars) is a more preferable selection since it allocates more space to the upper layers with bigger counters. However, as a trade-off, the smaller r hurts the mouse flow accuracy, as can be seen in Fig. 16(a), (c), and (e). To this end, we select the expansion factor $r = 4$ as the fixed parameter to balance the accuracy between the mouse and elephant flows.

Selection of d: With $r = 4$ (see the red bars in Fig. 16), we explore CL-MU's accuracy by varying the number of layers (d). As shown, CL-MU with $d = 3$ and $d = 4$ shows the best performance in the mouse flow estimation. Moreover, the accuracy of the mouse flows increases as more layers are defined, as shown in Fig. 16(a), (c), and (e). This trend reconfirms the effectiveness of our cross-layer update strategy. For the elephant flow estimation, CL-MU with $d = 3$

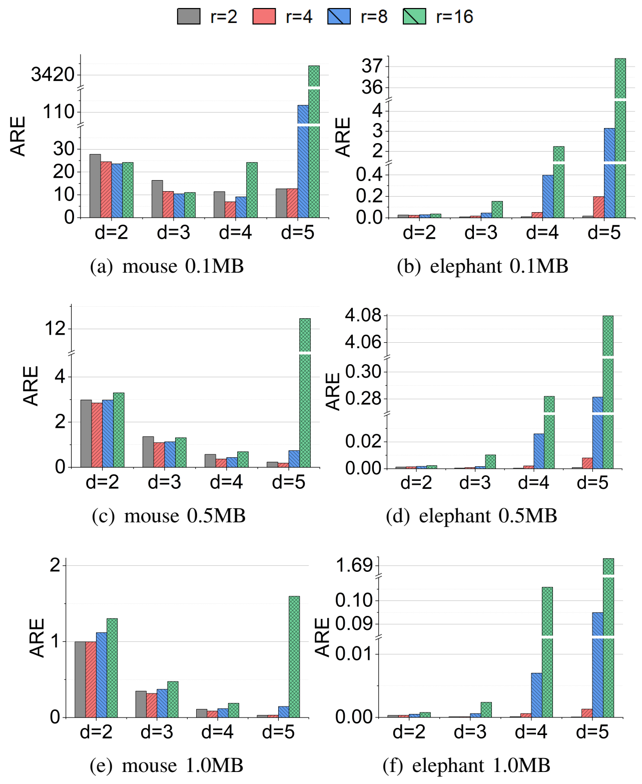


Fig. 16: AREs of mouse and elephant flows varying CL-MU's parameters: the number of layers (d from 2 to 5) and the expansion factor (r from 2 to 16). The results are based on a 5-second CAIDA trace (see section V-A) with three memory settings (i.e., 0.1 MB, 0.5 MB, and 1 MB).

outperforms $d = 4$ since more counters will be allocated to the higher layers with a smaller d. As such, $d = 3$ and $d = 4$ are both viable selections for CL-MU with a trade-off between the mouse and elephant flows' accuracy. In this work, we fixed the number of layers as $d = 3$ for the following reasons. First, for a switch's data plane, sketches need to be lightweight considering the constrained computation resources in the data plane. Therefore, CL-MU with $d = 3$ that saves one layer operations (i.e., memory access and compare operations) compared to $d = 4$ fits better. Second, current efforts of in-network detection pay more attention to the volumetric attacks (i.e., elephant flows) [4], [5], [35], [39], [65]. Thus, an elephant-friendly setting ($d = 3$) is more preferable in such scenarios.