Empowering Sketches with Machine Learning for Network Measurements*

Tong Yang†, Lun Wang†, Yulong Shen#, Muhammad Shahzad$, Qun Huang*

Xiaohong Jiang@, Kun Tan‡, Xiaoming Li†.

†Peking University. $North Carolina State University. #Xidian University. *Institute of Computing Technology at Chinese Academy of Sciences. @Future University Hakodate, ‡HUAWEI Technologies, Co. LTD

ABSTRACT

Network monitoring and management require accurate statistics of a variety of flow-level metrics, such as flow sizes, top-*k* flows, and number of flows. Arguably, the most commonly used data structure to record and measure these metrics is the sketch. While a significant amount of work has already been done on sketching techniques, there is still a lot of room for improvement because the accuracy of existing sketches depends a lot on the nature of network traffic and varies significantly as the network traffic characteristics change. In this paper, we propose the idea of employing machine learning to reduce this dependence of the accuracy of sketches on network traffic characteristics and present a *generalized machine learning framework* that increases the accuracy of sketches significantly. We further present three case studies, where we applied our framework on sketches for measuring three well-known flow-level network metrics. Experimental results show that machine learning helps decrease the error rates of existing sketches by up to 202 times.

CCS CONCEPTS

• **Networks → Network measurement**; • **Computing methodologies → Machine learning approaches**;

KEYWORDS

Network Monitoring, Machine Learning

ACM Reference Format:

Tong Yang†, Lun Wang†, Yulong Shen#, Muhammad Shahzad$, Qun Huang* Xiaohong Jiang@, Kun Tan‡, Xiaoming Li†.. 2018. Empowering Sketches with Machine Learning for Network Measurements. In *NetAI'18: ACM SIGCOMM 2018 Workshop on Network Meets AI ML , August 24, 2018, Budapest, Hungary*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3229543.3229545

1 INTRODUCTION

Network monitoring and management tasks such as traffic engineering [12, 16, 19–21, 27, 28], anomaly detection [3, 4, 17, 22], and forensics [26] require accurate and timely statistics of a variety of network flow-level metrics. One of the most effective methods to measure such flow-level metrics is to use *sketching* techniques. A sketching technique consists of two entities, a *sketch*, which is a set of counters or bitmaps associated with hash functions, and an *algorithm*, which is a set of simple operations that record approximate information about the metric of interest into the sketch. The *algorithm* can also estimate the metric of interest by applying appropriate statistical techniques on the *sketch*. Sketching techniques have found a widespread use in a variety of network monitoring and management tasks such as estimation of flow sizes [6], heavy hitters [3, 5, 22], and number of flows [13]. The key reason behind such widespread usage of sketches is that sketching techniques enable network administrators to do a provable trade-off among the accuracy of estimates, the memory used to store the sketch, and the computational overhead.

While researchers have made significant contributions in designing sketching techniques, we argue that a significant room for improvement still exists because the accuracies of existing sketches vary a lot with changing characteristics of network traffic [7]. The reason behind this is that the algorithms that insert approximate information about the desired metric into the sketches and then estimate that metric when required, are based on hand-derived theoretical models. In order to hand-derive such models, it is imperative to make simplifying assumptions about the network traffic because hand-deriving a model that covers all practical scenarios is just not possible. As the characteristics of network traffic vary a lot in practice and the simplifying assumptions do not always hold, the accuracies of existing sketches vary a lot when used in real-world deployments.

In this paper, we explore the idea of using machine learning to automate the process of building these models and embedding this automated process into the algorithm such that a human is no more required to foresee various network traffic characteristics and hand-derive the corresponding models. We propose a generalized machine learning framework that automatically and intrinsically

*Co-primary authors: Tong Yang and Lun Wang. Lun Wang finished this work under the guidance of his supervisor: Tong Yang. Corresponding author: Yulong Shen. This work is supported by Primary Research & Development Plan of China (2016YFB1000304), National Basic Research Program of China (973 Program, 2014CB340405), NSFC (61672061), National Science Foundation grants CNS 1616317 and CNS 1616273.

NetAI'18, August 24, 2018, Budapest, Hungary

ACM ISBN 978-1-4503-5911-5/18/08…$15.00

https://doi.org/10.1145/3229543.3229545

adapts the estimation algorithms of sketches by learning the current characteristics of network traffic on the fly, which in turn improves the accuracy of the sketches irrespective of any changes in network traffic characteristics. More specifically, instead of using statistical techniques that are based on hand-derived models, we continuously retrain machine learning models using a very small number of samples from the same traffic whose information is being stored in the sketch. As the samples are drawn from the same network environment as the traffic being monitored, they follow the same distribution as the traffic. Consequently, the machine learning models continue to adapt to any variations in the network traffic characteristics without requiring to first manually foresee the scenarios and then manually design statistical techniques for those scenarios. When estimating the desired flow-level metric, our framework employs these continuously adapting models, which leads to a significant increase in the accuracy.

2 BACKGROUND

2.1 Sketches to Estimate Flow Size

There are three classic sketching techniques used in recording the sizes of flows: count-min (CM) sketch [6], conservative-update (CU) sketch [11], and counter sum estimation method (CSM) sketch [18]. Their data structures are the same: each of them consists of d arrays of counters, and each array has w counters, as shown in Figure 1. We represent the i^{th} array of each sketch with $\mathbf{A}_i$ and the j^{th} counter of this i^{th} array with $A_i[j]$. Each array $\mathbf{A}_i$ is associated with a hash function $h_i(.)$.

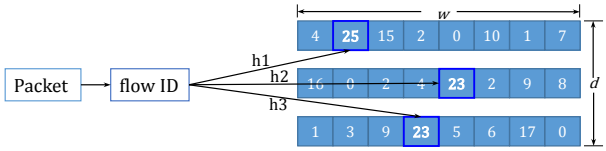

Figure 1: Structure of CM, CU, and CSM sketches.

Recording: The algorithm first obtains the flow ID f of the packet and computes the d hash functions $h_i(f)$, which map f to d counters $A_0[h_0(f)\%w]...A_{d-1}[h_{d-1}(f)\%w]$. Then the CM sketch increments all hashed counters by 1; the CU sketch increments the smallest hashed counter(s); the CSM sketch increments one of the hashed counters randomly.

Querying: To respond to a query with ID f, the sketching technique computes the d hash functions and retrieves the d counters. The CM and CU sketches report the value of the smallest counter as the size of the flow. The CSM sketch sums the values of the d hashed counters and subtracts a noise value from it.

2.2 Sketches to Estimate Top-k Flows

Top-k flows refers to the problem of identifying the k flows with the largest number of packets, and estimating the sizes of each of these top-k flows. The commonly used approach to recording top-k flows is a CM sketch with a min-heap [5]. Each node in the min-heap has two fields: a flow ID and a counter.

Recording: The algorithm inserts the packet with flow ID f into the CM sketch. If the value of the smallest counter is larger than the counter in the root node, the algorithm checks if f is in the min-heap. If the answer is yes, the algorithm increments the counter of that node by 1; otherwise, it deletes root node and inserts a new node with flow ID f and sets its counter equal to the estimate of the size of f calculated by the CM sketch.

Querying: To answer a query about the top-k flows, the algorithm returns all flow IDs in the min-heap along with their corresponding counter values.

2.3 Sketches to Estimate the Number of Flows

In this paper, we focus on the optimization of the FM sketch for flow number estimation. The FM sketch is comprised of d bitmaps. Each bitmap has w bits. As shown in Figure 2, we represent the i^{th} bitmap with $\mathbf{A}_i$ and the j^{th} bit of this bitmap with $A_i[j]$. Each bitmap $\mathbf{A}_i$ is associated with an independent hash function $h_i(.)$. Specifically, the hash function used in the FM sketch $h_i(.)$ maps half of all flow IDs to bit 0 (*i.e.*, LSB) of the i^{th} array, a quarter to bit 1, and so on.

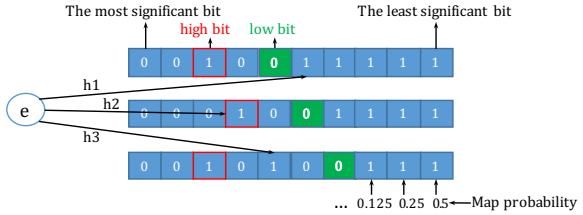

Figure 2: Structure of the FM sketch.

Recording: For each arriving packet, the algorithm of FM sketch computes the d hash functions $h_i(f)$ and sets the bits $A_i[h_i(f)\%w]$ to 1.

Querying: Let L_i represent the position of the rightmost zero in the i^{th} array. We call L_i the low-bit of the i^{th} array. To answer the query about the number of flows at any point in time, the algorithm of the FM sketch returns $1.2928 \times 2^{\frac{1}{d}\sum_{i=0}^{d-1} L_i}$ as an estimate.

There are also many other sketch works for network measurements, please refer to the literature [8–10, 23–25].

3 MACHINE LEARNING FRAMEWORK

3.1 Sampling

As the rate at which the traffic passes through a measurement point (such as a switch) can be very fast, the sketches for flow-level metrics become "full" in seconds, and have to be sent to a remote server for storage. These remotely stored sketches are then used for answering queries. To incorporate the characteristics of network traffic that generated any given sketch, we employ sampling and use only a small percentage of all packets to generate the models. Let $\mathcal{P}$ represent the set of flow IDs of all packets that generate a given sketch and let $\mathcal{S}$ represent the set of flow IDs sampled from $\mathcal{P}$. Note that if n sampled packets have a flow ID f, then that flow ID will appear in the sampled set $\mathcal{S}$ n times. In other words, the sampled set $\mathcal{S}$ is essentially a multi-set. In our framework, the network manager can specify the sampling rate based on the available resources. To keep our implementation fast, we choose to sample packets, not flows. To achieve a sampling rate of 1 in λ packets, we save the header of one packet after every $\lambda - 1$ packets, irrespective of packet's flow ID. A higher sampling rate, obviously, leads to higher estimation accuracy.

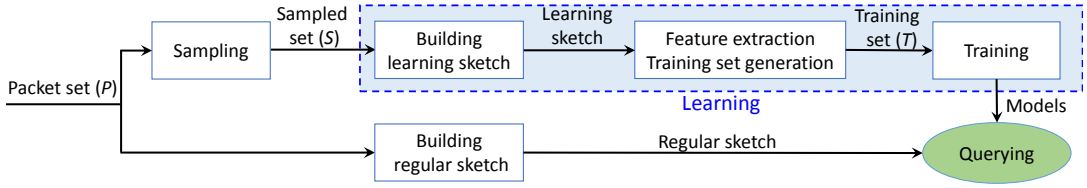

Figure 3: Block diagram of our generalized framework to augment sketches with machine learning.

3.2 Machine Learning

Next, we build a relatively small sketch, using only the packets in $\mathcal{S}$. We call the resulting sketch *learning* sketch, as shown in block "Building learning sketch" in Figure 3. To build the learning sketch, there are two options: (1) build it at the switch and send it to the server along with the regular sketch, (2) send packet headers directly to server and build learning sketch there. Both approaches are fine and the choice depends on the available computational resources at the switch and the bandwidth between the server and the switch.

Next, as shown in block "Feature Extraction, Training set generation" in Figure 3, we extract appropriate features from the learning sketch to build the adaptively build the theoretical model. The features that we extract depend on the flow-level metric being measured, and naturally are different for different metrics. In the next section, we will describe these features for three metrics. The ground truth for training comes from $\mathcal{S}$.

Using the learning sketch and training samples, we train a machine learning based theoretical model, as shown in block "Training" in Figure 3. As the training samples are drawn from the same network environment as the traffic being monitored and the learning sketch is built using these samples, they follow the same distribution as the whole traffic. As a result, the machine learning models reflect a mapping from the current traffic distribution to the metrics of interest, hiding all the complicated logics in the parameters of the model. We argue that although we need to design features and machine learning model for every kind of sketch, this framework still significantly reduces the designers' labor. The designers do not need to worry about different network environments any more. Furthermore, designing features and machine learning model is usually just picking from what features and models you have, which is much simpler than designing a new sketch algorithm. Finally, as shown in block "Querying" in Figure 3, using the regular sketch along with this machine learning based theoretical model, the framework is now ready to answer any queries. The block "Querying" in Figure 3 represents the process of answering the queries. Next, we present three case studies to show how we apply our framework to existing sketches.

4 CASE STUDIES

4.1 Estimating Flow Sizes

For flow size estimation, we started with the intuition that the flows for which the estimation error is already acceptably small do not need further improvements. It is the flows for which the estimation error is high that need further improvement and where machine learning can help. We name such flows that can experience high estimation error as *error-prone* flows. We observed that for the majority of such flows, the smallest counter value v_1 is much smaller than the second smallest counter value v_2. We only do the machine learning optimization for error-prone flows.

First, we generate a learning sketch using the packets in the sampled set $\mathcal{S}$. Next, we traverse through all flow IDs in the hash table and identify the error-prone flows by comparing the absolute difference between the values of two smallest hashed counters of each flow. Then, we use them for training by using each error-prone flow as a training sample. From our extensive tests on real traces, we found that the actual size of any given flow is almost a linear combination of the hashed counters. Therefore, for any given error-prone flow, the values of its d hashed counters in the learning sketch serve as the features and the actual values of the packet counts of the flow, recorded in the hash table, serve as target. We choose linear regression as our machine learning algorithm. The advantage of linear regression is its very low computational and space complexity. Because we observe in Section 6 that a trained model can be used for a long time, a more delicate model like support-vector machine [15] or neural network [14] can be used. We will conduct related researches in our future work.

To respond to a query requesting an estimate of the current size of a flow, we first check whether this flow is error-prone flow by comparing the absolute difference between the values of its two smallest hashed counters in the regular sketch with the threshold θ. If the flow is not an error-prone flow, then we use the conventional algorithm of the sketching technique to estimate the size of the flow. If the flow is an error-prone flow, then we estimate its size by applying the trained linear regressive model on the values of the d hashed counters of this flow in the regular sketch.

4.2 Estimating Top-k Flows

Top-k flow estimation suffers from two types of errors, *i.e.*, estimation error and misclassification error. Estimation error in top-k flow problem is the same as that in the flow size estimation problem. The misclassification error occurs primarily due to the over-estimation error of the CM sketch, which causes some flows that actually do not belong to the top-k flows to be mistakenly inserted into the min-heap. For the sake of presentation, in this section, we call such flows *mice flows*.

To reduce the misclassification error, we leverage the fact that after the ID of a mice flow along with its currently estimated size is inserted into a min-heap, its counter is rarely incremented compared to the flows that are correctly identified as among the top-k flows and entered into that min-heap. Consequently, if we can keep track of how many times the counter of each flow in the min-heap is incremented after it is inserted, we can identify the flow IDs in

the min-heap that are of mice flows. Next, we describe how we apply our machine learning framework to this problem.

We use all flow IDs in the learning min-heap to generate training sets for both classification and estimation tasks. However, for each task, we use different features. For *classification task*, we choose to use the values of the d hashed counters of each flow in the learning CM sketch and the difference between the initial counter field and the counter field in the learning min-heap as features, and the ground truth whether this flow ID is correctly inserted into the learning min-heap or not as class label. For *estimation task*, we use the values of the d hashed counters in the learning CM sketch as the features, and use the actual size of that flow as target. We use logistic regression as our machine learning algorithm for the classification task and linear regression for the estimation task.

To answer a query about the top-k flows, for each flow with ID f in the regular min-heap, we estimate its size by applying logistic regression to first identify and eliminate any erroneously inserted flow IDs, and then estimate the sizes for the remaining flow IDs using linear regression model. Finally, we return the flow IDs remaining in the min-heap and the corresponding estimated size.

4.3 Estimating Number of Flows

The FM sketch is accurate only when d is large for estimating the number of flows. Unfortunately, that requires a large amount of high-speed memory. This is one of the biggest shortcomings of FM sketch. With the help of machine learning, we aim to achieve the required accuracy using a smaller value of d.

Unlike the sketches discussed until now, a single learning FM sketch only provides one training sample. Consequently, instead of generating a single learning FM sketch from sampled set $\mathcal{S}$, we first create multiple subsets of the sampled set $\mathcal{S}$, and then generate a learning FM sketch from each of those subsets. From each learning FM sketch, we use the d locations, L_i, of low-bits and another d locations, H_i, of high-bits (a high-bit H_i represents the position of the leftmost one in the i^{th} array) as features. The target is the actual exponent part of the query formula of FM sketch. We have already seen in Section 2.3 that the location of low-bits is a function of the number of flows, and can thus be used as features. The motivation behind using the locations of high-bits as features is similar: the position of the high-bit is also a monotonically increasing function of the number of flows.

To answer the query about the number of flows at any point in time, we apply the linear regressive model on the values of d locations of low-bits and another d locations of high-bits in the given regular FM sketch and get the exponent part of the query formula. After that, we use the estimation formula of FM sketch to estimate of number of flows.

5 IMPLEMENTATION

As shown in Figure 4, there are two parts in our architecture: a switch and a server. The switch is responsible for generating the *regular* sketches and producing the sampled set $\mathcal{S}$.

The server implements and runs the machine learning aspects and also receives, processes, and responds to the queries. We use three threads to accomplish these tasks, each of which is bound with a CPU core. To receive packets from the switch, we use DPDK [2]. The server contains a memory pool, which is a data structure defined and managed by DPDK and used to store the packets received from NIC (*i.e.*, flow IDs of the sampled packets as well as the regular sketches). Furthermore, there are three circular queues (also called rings). One of these three rings is a hardware (HW) ring dedicated for receiving packets from NIC, while the other two are software (SW) rings acting as a pipe to transfer data. These rings are also data structures defined and managed by DPDK, and are used to store pointers to the data in the memory pool.

As soon as the switch sends any packet (containing either flow IDs of sampled packets or a regular sketch), the network interface card (NIC) stores it in the memory pool and inserts its address-pointer into the HW ring. We have implemented a thread that is bound with a CPU core polling the HW ring for packets' arrival. Let us call it thread 1. As soon as it receives an address pointer from the HW ring, it retrieves the corresponding packet from the memory pool and analyses its contents. If the packet payload is composed of flow IDs, then the thread 1 inserts its address pointer into SW ring 1. If it contains a regular sketch, then the thread 1 inserts its address pointer into SW ring 2.

We have implemented a second thread that is bound with a second CPU core polling the SW ring 1 for packets coming from thread 1. Lets call it thread 2. As soon as it receives an address pointer from the SW ring 1, it retrieves the corresponding packet from the memory pool and based on the sketching technique being tested, it uses flow IDs in that packet payload to generate learning sketches, which is then used to train machine learning models specific to that sketching technique. Every time this thread creates a new machine learning model, it passes that model to a third thread, called thread 3, which is bound with a third CPU core. In addition to receiving the trained model from thread 2, thread 3 also polls the SW ring 2 for packets coming from thread 1. As soon as it receives an address pointer from the SW ring 2, it retrieves the corresponding packet from the memory pool and uses the regular sketch in that packet along with the model it received from thread 2 to answer any queries with higher accuracy.

6 EXPERIMENTAL RESULTS

Computer Settings: We used two servers, both equipped with 2 Intel CPUs (Xeon E5-2630, v2, 2.60 GHz, 12 physical cores) and 80GB memory, two network interface cards (Intel I340T2/82580, 1Gbps), running Ubuntu 14.04.3 LTS. The first server used DPDK to send the captured traces from switch-machine to the other server. This emulates the packets being captured and processed at line-rate.
Traffic Traces: We collected real network traffic traces from a tier-1 router. We identify flows using the standard 5-tuple. We also generate synthetic traffic traces containing 10M packets each with varying skewness according to zipfian distributions. We will use more real-world datasets like CAIDA [1] in our future work to further demonstrate the effectiveness of our framework.
Evaluation Metrics: We evaluate the accuracy of sketches in terms of average absolute error (**AAE**), average relative error (**ARE**), and **AAE ratio**. Let r_i represent the true size of the i^{th} flow, $\hat{r}_i$ represent the estimated size of that flow, and n represent the total number of flows. Then, AAE is defined as $\frac{1}{n}\sum_{i=1}^{n}|\hat{r}_i - r_i|$, and ARE is defined

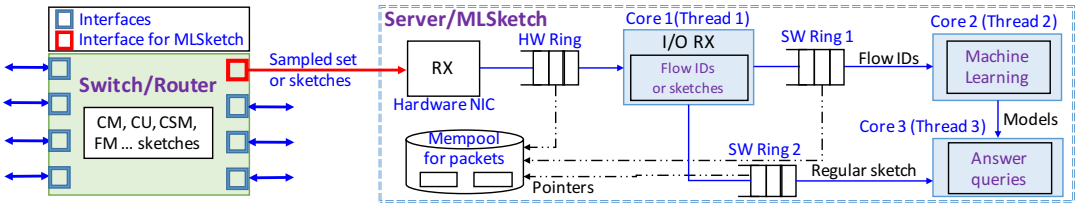

Figure 4: Block diagram of the implementation of our machine learning framework.

as $\frac{1}{n}\sum_{i=1}^{n}\frac{|\hat{r}_i - r_i|}{r_i}$. AAE highlights the accuracy of sketches on large flows, while ARE highlights the accuracy on small flows. AAE ratio is defined as the ratio of the AAE achieved by the sketch with machine learning to the AAE achieved by the sketch without machine learning.

Parameters Selection: In all our experiments, we allocate a fixed 1 Mbit memory to all sketches. We chose to use $d = 4$ in all our experiments. The size of each counter for sketches is 16 bits. We measured the AAE achieved by CM, CU, and CSM augmented with machine learning with sampling rates varying from 1 in 5 packets to 1 in 1000 packets. We observe that the improvement by increasing the sampling rate from 1 in 5 packets to 1 in 100 packets is very nominal. Therefore, in all our subsequent experiments, we use a sampling rate of 1 in 100 packets.

6.1 Evaluation for Flow Size Estimation

Training Frequency: Next, we determine the frequency at which one should retrain the machine learning model for optimum improvement in accuracy. Figure 5 plots the AAE ratio of "training once" and "training always" over regular CM sketch. We observe that training on a decent amount of data collected is enough to give good accuracy for the next hour. Consequently, in practice, the machine learning based training will only be an infrequent process and will not use a large amount of CPU, bandwidth, and memory of a switch/router.

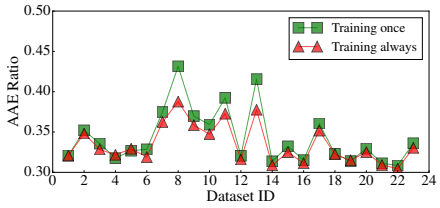

Figure 5: Training once vs. training always.

Evaluation with CM Sketch As shown in Figure 6, the AAE ratio of CM sketch with machine learning across individual flows lies in the range [2.65, 3.31] with a mean of 2.95.

Evaluation with CU Sketch As shown in Figure 7, the AAE ratio of CU sketch with machine learning across individual flows lies in the range [2.04, 2.22] with a mean of 2.13.

Evaluation with CSM Sketch As shown in Figure 8, the AAE ratio of CSM sketch with machine learning on different datasets lies in the range [10.68, 12.42] with a mean of 11.47.

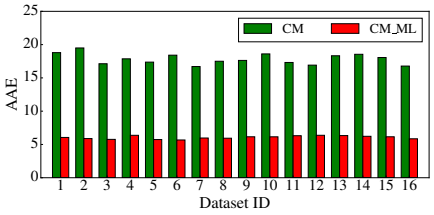

Figure 6: AAE of CM sketches

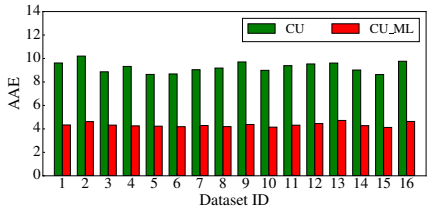

Figure 7: AAE of CU sketches

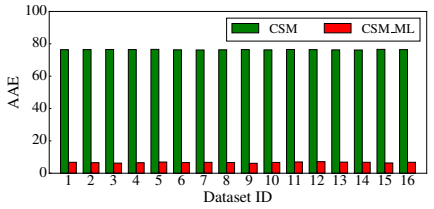

Figure 8: AAE of CSM sketches

6.2 Evaluation for Top-k Flow Estimation

Our experimental results show that our classification scheme correctly identifies over 80% of flows that are actually not among top-k flows but are incorrectly inserted into the min-heap. We observed from our experiments that when using different 10-minute network traces, the number of such flows erroneously inserted into the min-heap lied in the range of 8 to 27. Figure 9 plots the average of the relative errors between the actual sizes of such flows and their corresponding sizes stored in the min-heap. We observe from this figure that these AREs are very large, showing that these flows indeed are not top-k flows.

Our experimental results show that the AREs of CM_Heap_ML for the top 1000 flows are 14.89 to 202.15 times smaller than the AREs of CM_Heap for the corresponding flows with a mean of 75.36. Our experimental results also show that the AAEs of CM_Heap_ML for the top 1000 flows are 1.11 to 2.59 times smaller than the AAEs of

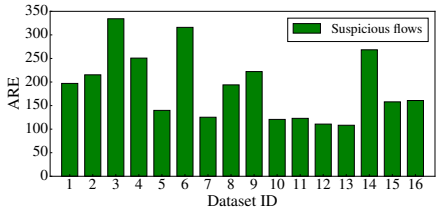

Figure 9: ARE of incorrectly inserted flows

CM_Heap for the corresponding flows with a mean of 1.75. Figures 10 and 11 plot the AREs and AAEs, respectively, of CM_Heap and CM_Heap_ML.

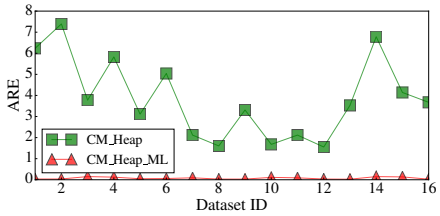

Figure 10: ARE for the Top-k flows

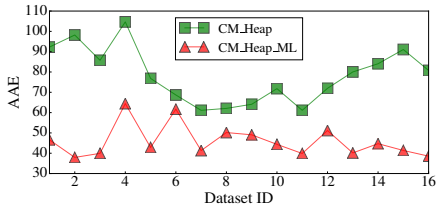

Figure 11: AAE for the Top-k flows

6.3 Evaluation for Flow Number Estimation

Our experimental results show that the relative errors (REs) of the FM_ML sketch are up to 1128.4 times smaller than the corresponding REs of FM sketch with a mean of 117. Figure 12 plots the REs of FM and FM_ML sketch in 10 randomly chosen such traces of 10M packets. We clearly observe from this figure that machine learning significantly reduces the RE of the FM sketch.

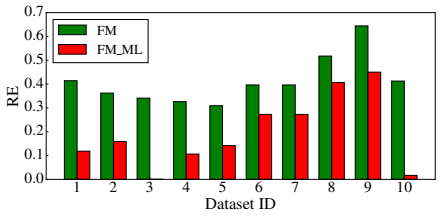

Figure 12: RE comparison

7 CONCLUSION

In this paper, we proposed a *machine learning framework* to reduce the dependence of sketches on network traffic characteristics, which in turn improves their accuracy. We take several sketches that are currently used to estimate flow sizes, top-k flows, and number of flows and apply our framework to them. Through extensive evaluation, we showed that our generalized machine learning framework enables us to reduce the error rates of existing sketches by up to 202 times. We hope that this work would spark more research in the area of automating the sketching techniques.

REFERENCES

[1] Caida. http://www.caida.org/home/.
[2] DPDK websit. http://dpdk.org/.
[3] R. Ben Basat, G. Einziger, R. Friedman, M. C. Luizelli, and E. Waisbard. Constant time updates in hierarchical heavy hitters. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 127–140. ACM, 2017.
[4] C. Callegari and N. Cyprus. Statistical approaches for network anomaly detection. In *Proc. ICIMP*, 2009.
[5] G. Cormode. Sketch techniques for approximate query processing. *Foundations and Trends in Databases. NOW publishers*, 2011.
[6] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.
[7] G. Cormode and S. Muthukrishnan. Summarizing and mining skewed data streams. In *Proc. SDM*, 2005.
[8] H. Dai, L. Meng, and A. X. Liu. Finding persistent items in distributed, datasets. In *Proc. IEEE INFOCOM*, 2018.
[9] H. Dai, M. Shahzad, A. X. Liu, and Y. Zhong. Finding persistent items in data streams. *Proceedings of the VLDB Endowment*, 10(4):289–300, 2016.
[10] H. Dai, Y. Zhong, A. X. Liu, W. Wang, and M. Li. Noisy bloom filters for multi-set membership testing. In *Proc. ACM SIGMETRICS*, pages 139–151, 2016.
[11] C. Estan and G. Varghese. New directions in traffic measurement and accounting. *ACM SIGMCOMM*, 2002.
[12] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul. Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags. In *NSDI*, volume 14, pages 533–546, 2014.
[13] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *Journal of computer and system sciences*, 31(2):182–209, 1985.
[14] S. Haykin and N. Network. A comprehensive foundation. *Neural networks*, 2(2004):41, 2004.
[15] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf. Support vector machines. *IEEE Intelligent Systems and their applications*, 13(4):18–28, 1998.
[16] Q. Huang, X. Jin, P. P. Lee, R. Li, L. Tang, Y.-C. Chen, and G. Zhang. Sketchvisor: Robust network measurement for software packet processing. In *SIGCOMM 2017*.
[17] Q. Huang and P. P. Lee. Ld-sketch: A distributed sketching design for accurate and scalable anomaly detection in network data streams. In *IEEE INFOCOM 2014*.
[18] T. Li, S. Chen, and Y. Ling. Per-flow traffic measurement through randomized counter sharing. *ToN*, 2012.
[19] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman. One sketch to rule them all: Rethinking network flow monitoring with univmon. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 101–114. ACM, 2016.
[20] C. Monsanto, J. Reich, N. Foster, J. Rexford, D. Walker, et al. Composing software defined networks. In *NSDI*, volume 13, pages 1–13, 2013.
[21] M. Moshref, M. Yu, R. Govindan, and A. Vahdat. Trumpet: Timely and precise triggers in data centers. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 129–143. ACM, 2016.
[22] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford. Heavy-hitter detection entirely in the data plane. In *Proceedings of the Symposium on SDN Research*, pages 164–176. ACM, 2017.
[23] K. Xie, X. Li, X. Wang, and et al. Fast tensor factorization for accurate internet anomaly detection. *IEEE/ACM Transactions on Networking*, 25(6):3794–3807, 2017.
[24] K. Xie, X. Li, X. Wang, and et al. On-line anomaly detection with high accuracy. *IEEE/ACM Transactions on Networking*, 2018.
[25] K. Xie, L. Wang, and et al. Accurate recovery of internet traffic data: A sequential tensor completion approach. *IEEE/ACM Transactions on Networking (TON)*, 26(2):793–806, 2018.
[26] Y. Xie, V. Sekar, D. A. Maltz, M. K. Reiter, and H. Zhang. Worm origin identification using random moonwalks. In *Proc. IEEE SPSR*, 2005.
[27] M. Yu, L. Jose, and R. Miao. Software defined traffic measurement with opensketch. In *Proc. USENIX NSDI*, pages 29–42, 2013.
[28] Y. Zhang. An adaptive flow counting method for anomaly detection in sdn. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*, pages 25–30. ACM, 2013.