Enhanced Machine Learning Sketches for Network Measurements

Hengrui Wang, Huiping Lin, Zheng Zhong, Tong Yang, and Muhammad Shahzad

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 current best technique to measure these metrics is sketches. 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 varies with changing characteristics of network traffic. In this paper, we propose the idea of using machine learning to improve the accuracy of sketches, and propose a *generic machine learning framework* to reduce the dependence of accuracy of sketches on network traffic characteristics. We further present three case studies, where we applied our machine learning framework on sketches for measuring three flow-level network metrics, namely flow sizes, top-k flows, and number of flows. We implemented and extensively evaluated this framework for these three metrics using both real-world and synthetic traffic traces. To the best of our knowledge, this is the first work that uses machine learning to reduce the dependence of sketching techniques on the characteristics of network traffic. We have released all our traces and implementation codes at Github.

Index Terms—Network measurements, sketch, machine learning

1 Introduction

1.1 Motivation

Network monitoring and management tasks such as traffic engineering [21], [22], [30], [39], [45], anomaly detection [32], [40], [43], and sharing web caches [16] require accurate and timely statistics of a variety of network flow-level metrics such as flow sizes [6], [9], heavy hitters [3], [7], number of flows [19], heavy changers [24], and several others. These flow-level metrics often have to be measured from high speed network traffic, which cannot be stored for off-line analysis due to its volume. 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 comprised of a set of counters or bitmaps associated with hash functions, and an *algorithm*, which is comprised of a set of simple operations that record approximate information about the metric of interest from each flow into the sketch using a small amount of memory. At any given point, during or after recording the information, the *algorithm* can also estimate the metric of interest of any target flow by applying appropriate statistical techniques on the *sketch*. Sketching techniques have found a widespread adoption in a variety of network monitoring and management tasks such as estimation of flow sizes [9], [15], heavy hitters [7], number of flows [19], [20], heavy changers [24], packet delay [34], latency [37], and persistent items [25]. The two key reasons behind such widespread adoption are that sketching techniques enable network administrators to 1) estimate the metric of interest with a bounded error, and 2) do a provable trade off among the accuracy of the estimation, the memory used to store the sketch, and the computational overhead.

While researchers have made significant contributions in designing sketching techniques, and those techniques work well in specific scenarios, we argue that a significant room for improvement still exists because the accuracy of existing sketches may degrade with changing characteristics of network traffic. The reason behind this is that the information stored in sketches is only approximate and existing sketches estimate the amount of noise in the stored information and remove it when answering any query. The idea of estimating the amount of noise is intuitive. To model this noise, several existing sketches make assumptions about network traffic characteristics and use a single theoretical model to quantify the noise. However, practically, the amount of noise depends on the distributions of various characteristics of network traffic and the assumptions do not always hold. Consequently, as the characteristics of network traffic always vary over time, existing sketches can not always efficiently model noises. So the accuracies of existing sketches also vary. [5], [10]. Unfortunately, in the current practice of designing the sketching algorithms, it is impossible to eliminate such assumptions about the characteristics of network traffic because these assumptions are required to make the

• *Hengrui Wang, Huiping Lin, and Zheng Zhong are with the School of Computer Science, National Engineering Laboratory for Big Data Analysis Technology and Application, Peking University, Beijing 100871, China. E-mail: {wanghr1230, phoenixrain, billyzhong}@pku.edu.cn.*
• *Tong Yang is with the School of Computer Science, National Engineering Laboratory for Big Data Analysis Technology and Application, Peking University, Beijing 100871, China, and also with Peng Cheng Laboratory, Shenzhen, Guangdong 518066, China. E-mail: yangtongemail@gmail.com.*
• *Muhammad Shahzad is with North Carolina State University, Raleigh, NC 27695 USA. E-mail: mshahza@ncsu.edu.*

Manuscript received 16 August 2020; revised 6 March 2022; accepted 3 May 2022. Date of publication 1 July 2022; date of current version 13 March 2023.
This work was supported in part by the Key-Area Research and Development Program of Guangdong Province under Grant 2020B0101390001 and in part by the National Natural Science Foundation of China (NSFC) under Grants U20A20179 and 61832001.
(Corresponding author: Tong Yang.)
Recommended for acceptance by E.J. Kim.
Digital Object Identifier no. 10.1109/TC.2022.3185560

theoretical development of statistical techniques tractable. One candidate solution to overcome the challenges introduced by the changing network traffic characteristics is the following three-step approach: 1) foresee all possible scenarios that can occur in practice, 2) make appropriate assumptions for each scenario, and 3) develop a dedicated statistical technique to estimate that metric for each scenario. Unfortunately, this three-step approach has two major shortcomings. First, it is not always possible to foresee all possible scenarios that can occur in practice. Second, manually developing statistical models for each scenario is a laborious and time-consuming task, and it is not always possible to derive a closed form or practically usable statistical technique for each scenario.

1.2 Our Solution

In this paper, we explore how to improve the accuracy of sketching algorithms by reducing their dependence on the characteristics of network traffic. *By achieving this objective, we envision that the practice of designing sketching techniques will fundamentally change for better because researchers can then shift their attention from foreseeing all scenarios and individually handling them to increasing the accuracy of sketching algorithms and reducing the memory footprints of the sketches.* To achieve this objective, we propose the idea of using machine learning and propose a generic machine learning framework that learns the current characteristics of network traffic and adapts the estimation algorithms of sketches to those characteristics on the fly, which in turn improves the accuracy of the sketches. To further improve the estimation accuracy, we classify flows into different clusters and train each cluster an estimation model.

More specifically, instead of using statistical techniques, we continuously train machine learning models using a very small number of samples from the same traffic whose information is stored in a relatively "small" sketch, to dynamically remove the appropriate amount of noise. Because the models are trained using only a small fraction of packets in the traffic rather than all packets, we keep the computational cost of training small and amenable for implementation on commodity hardware. When estimating the flow-level metric, we use these models to obtain the estimate. As these models are trained using samples from the same traffic from which the sketch was built, it automatically learns and adapts to the characteristics of that traffic. Consequently, the same sketching technique gives very high accuracy under all types of scenarios, without requiring to manually foresee all the scenarios and then manually design statistical techniques for them. *In addition, as we train machine learning models continuously, our model will be frequently updated to handle workload changes.*

To validate the feasibility of using machine learning techniques for reducing the dependence of sketching algorithms on network traffic characteristics, we incorporated machine learning in the sketching techniques for estimating three well-known flow-level metrics, namely flow sizes, top-k flows, and the number of flows. More specifically, we augmented the statistical estimation techniques of the algorithms of these sketching techniques with models trained using

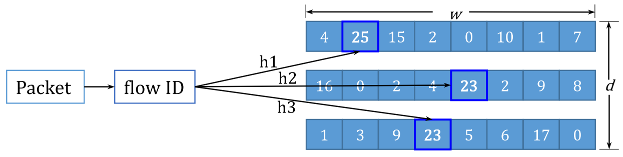

Fig. 1. Structure of CM and CU sketches.

machine learning. We then implemented and extensively evaluated these sketching techniques augmented with machine learning in a variety of scenarios. Our results show that machine learning helps decrease the error rates of existing sketches by orders of magnitude. Furthermore, the machine-learning-based sketching techniques show no notable deterioration of accuracy with changes in the characteristics of network traffic. These results show that machine learning has the potential to not only make sketching algorithms less dependent on the characteristics of network traffic but also open a new door for improving the accuracy and decreasing the memory footprint of sketches. To the best of our knowledge, this is the first work that aims at reducing the dependence of sketching techniques on the characteristics of network traffic and proposes to use machine learning to achieve this objective.

Contributions: In summary, we make following three key contributions in this paper.

- We introduce the idea of using machine learning to improve the accuracy of sketches. We present a generic framework to augment sketching techniques with machine learning by learning network traffic characteristics. We further optimize our framework by training different machine learning models for different flows. Our learned model will be periodically updated in case of characteristics changes.
- We present three case studies of our framework by applying machine learning to three typical kinds of sketches for three well-known flow-level metrics: flow sizes, top-k flows, and the number of flows.
- We implemented our framework and performed extensive experiments using real-world network traffic traces to evaluate the improvement in accuracy of existing sketches due to machine learning. Our results show that machine learning helps decrease the error rates of existing sketches by orders of magnitude.

2 Background and Related Work

2.1 Estimation of Flow Size

There are three well known classic sketching techniques that are frequently used in recording and estimating the sizes of flows: count-min (CM) sketch [9] and conservative-update (CU) sketch [15]. Their data structures are the same: each of them consists of d arrays of counters, and each array has w counters, as shown in Fig. 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 $A_i[j]$, where $0 \leqslant i < d$ and $0 \leqslant j < w$. Each array $\mathbf{A}_i$ is associated with a hash function $h_i(.)$. Before the algorithm of the given sketching technique starts recording information about the sizes of the flows, all counters are first initialized to 0.

Recording: To record information about the sizes of flows in the given sketch, on the arrival of each packet, the algorithm of that sketching technique first obtains the flow ID f from the packet header and then computes the d hash functions $h_i(f)$, where $0 \leqslant i < d$. The outputs of these d hash functions map the flow ID f to d counters $A_0[h_0(f)\%w]\ldots A_{d-1}[h_{d-1}(f)\%w]$. To make the subsequent discussion easy to follow, we name these d counters `hashed counters of` f. After this point, the algorithm of each sketching technique uses different rules to increment one or more of these hashed counters: the CM sketch increments all hashed counters by 1; the CU sketch increments only the smallest hashed counter(s).

*Querying:*To respond to a query requesting an estimate of the current size of a flow with ID f in a given sketch, the algorithm of the corresponding sketching technique first computes the d hash functions and retrieves the d hashed counters of f. The algorithms of the CM and CU sketches report the value of the smallest of the d hashed counters as the estimate of the current size of the flow f.

2.2 Estimation of Top-K Flows

Top-k flows refers to the problem of identifying the k flows that have the largest number of packets among all flows, and estimating the sizes of each of these top-k flows. The most commonly used approach to identify top-k flows and to estimate their sizes is to use a CM sketch with a min-heap [7], [9] According to this method, the algorithm first initializes all counters of a CM sketch to 0 and initializes a min-heap with k nodes. Each node in the min-heap has two fields: a flow ID and a counter. At initialization, the flow ID field of each node is empty and the value of each counter is 0.

Recording: On arrival of any packet with flow ID f, the algorithm first inserts it into the CM sketch, i.e., increments the d hashed counters of f by 1. After that, if the value of the smallest counter among its d hashed counters is larger than the counter in the root node of the min-heap, the algorithm checks whether the flow ID f is currently in the flow ID field of any node in the min-heap. If f is already in the flow ID field of a node in the min-heap, the algorithm increments the counter of that node by 1; otherwise, it replaces the value in the flow ID field of the root node with f and the counter in the root node with the current estimate of the size of f as calculated by the CM sketch, i.e., the current value of the smallest hashed counter of f. Next, the algorithm rearranges the nodes of the min-heap so that it becomes a valid min-heap. To speed up the process of checking whether a given flow ID is in the min-heap, one effective way is to use a hash table, in which the key is the flow ID and the value is the pointer of the node in the min-heap containing that flow ID. Every time a new flow ID is inserted into (or an existing flow ID is removed from) the min-heap, the corresponding entry in the hash table is updated.

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

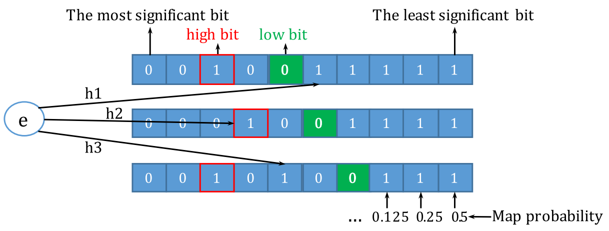

Fig. 2. Structure of the FM sketch.

2.3 Estimation of the Number of Flows

There are several common techniques that use sketches to estimate the number of flows, such as FM sketch [19], LogLog [14], and HyperLogLog [18]. In this paper, we focus on the optimization of FM sketch. Flajolet and Martin proposed an approximate counting algorithm that is extensively used to estimate the number of flows and is considered one of the best techniques for this task [7]. It is named FM sketch [19]. Similar to the sketches we have discussed until now, the sketch of FM sketch is also comprised of d arrays. However, instead of w counters per array, each array in this sketch has w bits. To stay consistent in notations, as shown in Fig. 2, we represent the i^{th} array of the FM sketch with $\mathbf{A}_i$ and the j^{th} bit of this i^{th} array (counting from right to left) with $A_i[j]$, where $0 \leqslant i < d$ and $0 \leqslant j < w$. Each array $\mathbf{A}_i$ is associated with an independent hash function $h_i(.)$. Unlike the hash functions used in sketches we have described until now, the hash functions used in the FM sketch do not have uniformly distributed output. More specifically, the hash function $h_i(.)$ 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. Formally, its distribution is defined as $P\{h_i(.) = j\} = 1/2^{(j+1)}$, where $0 \leqslant j < w$. Such hash functions are actually very easy to implement [7], [19].

Recording: To record information about the number of flows in the FM sketch, 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, where $0 \leqslant L_i < w$ and $0 \leqslant i < d$. We call L_i the low-bit of the ith 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 of the number of flows seen until that point in time. In Fig. 2, where $d = 3$ and $w = 10$, $L_1 = 5$, $L_2 = 4$, $L_3 = 3$, and the estimate of the number of flows is, thus, 20.68. To understand the logic behind the estimation formula of FM sketch mentioned above, we refer the interested reader to [19].

2.4 Other Prior Sketches and Key Limitations

In addition to the three sketching techniques we presented, several other sketching techniques have also been proposed to estimate flow sizes, such as Count sketch [6], CMM sketch [13], and Counter braids [29]. Similarly, other techniques for top-k flows include Count sketch with min-heap [6], Augment sketch [33], and others [11]. Additional techniques for estimation of number of flows include Bloom filters [8], distinct sampling [17], [20], and kMin [4].

Sketches also find applications in several other networking related tasks such as detecting heavy changers and DDoS attack [12], [23], [24], and latency measurement [34],

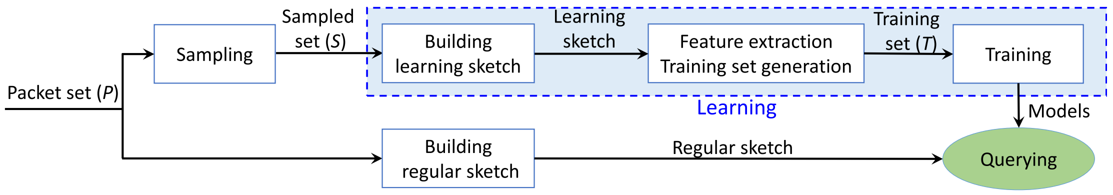

Fig. 3. Block diagram of our generic framework to augment sketches with machine learning.

[36]. Another direction of work on sketches, such as Open sketch [41], UnivMon [27], FlowRadar [26], SCREAM [31], RISC [28], and CSAMP [35], provide scalable system architectures or approaches to achieve flexible use of various sketches in a wide range of network functions.

2.5 Key Limitations of the Prior Sketches

The key limitation of all of these sketching techniques is that their accuracy varies with changing characteristics of network traffic. This happens because when answering queries, existing sketching techniques cannot accurately remove the noise from the measurements because the amount of noise is a function of various characteristics of the network traffic, and these characteristics are not known in advance. The design goal of this paper is to remove the noise as accurately as we can. As mentioned in Section 1, the shortcomings of the three-step approach makes it difficult to derive a closed form or practically usable statistical technique for each scenario. To address this issue, we propose a generic machine learning framework to dynamically and automatically learn the noise and remove it.

3 Machine Learning Framework

3.1 Rationale

As mentioned earlier, the information stored in sketches is only approximate. Consequently, when algorithm of a sketch estimates the value of that metric for any arbitrary flow, that estimated value is essentially comprised of two parts: 1) actual value of the metric, and 2) noise. Algorithms of existing sketches either simply do not try to remove this noise, such as in the cases of CM and CU sketches, or try to estimate the amount of noise and remove it before returning the answer. To quantify this noise, several existing sketches make assumptions about network traffic characteristics and use a single theoretical model to quantify the noise. However, practically, the amount of noise depends on the distributions of various characteristics of network traffic and the assumptions do not always hold. Consequently, as the characteristics of network traffic vary over time, the accuracies of existing sketches also show significant difference [5], [10].

Keeping this limitation of prior art in view, our objective is to design a framework that can make the accuracy of sketching algorithms largely independent of any assumptions about the characteristics of network traffic. The key intuition behind our proposed framework is that instead of using a single theoretical model, the sketching algorithms should learn the network traffic characteristics on the fly and use adaptive theoretical models to quantify the noise. The obvious candidates for an adaptive theoretical model are the machine learning techniques. Next, we describe the components of our proposed framework that enables sketching techniques to accurately measure flow-level metrics largely independent of the characteristics of network traffic.

3.2 The Framework

Fig. 3 shows the block diagram of our framework. *We should notice that all these processes are continuously executing in order to periodically update the model. Otherwise, the characteristic may change and make our learned model not work anymore.* The block ”Building regular sketch” represents the process of recording information into the sketch, which we have already described in detail in Section 2 using CM and CU sketches as examples. We name the sketch generated by this block *regular* sketch. Next, we describe the remaining blocks in the figure.

3.2.1 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. Every time a sketch is transferred to remote storage, a new sketch is initialized at the measurement point. To incorporate the characteristics of network traffic that generated any given sketch, one option is to use all packets when building a machine learning based adaptive theoretical model to accurately estimate the metric of interest. We represent the set of flow IDs of all packets that generate a given sketch by $\mathcal{P}$. Unfortunately, this can create a lot of overhead and the framework may not be able to keep up with line rate. The other option, inspired by tools such as sFlow [38], is to employ sampling and use only a small percentage of all packets to generate the models. We represent the set of flow IDs sampled from $\mathcal{P}$ by $\mathcal{S}$. 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 advocate saving the header of one packet after every $\lambda - 1$ packet, irrespective of the packet's flow ID. A higher sampling rate, obviously, leads to higher estimation accuracy.

3.2.2 Machine Learning

In our framework, next, we build a sketch that is relatively smaller compared to the ”Building regular sketch” block, using only the packets in sampled set $\mathcal{S}$. We name the resulting sketch *learning* sketch. To build the learning sketch, there are two options. The first option is to build it at the switch and send it to the server along with the regular sketch. The other option is to send packet headers directly to server and build learning sketch at the server. 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. The block "Building learning sketch" in Fig. 3 represents the process of building this learning sketch.

After generating the learning sketch, in our framework, we extract appropriate features from the learning sketch to build the adaptive theoretical model. The features that we extract depend on the flow-level metric being measured, and naturally are different for different metrics. The ground truth for training comes from the set $\mathcal{S}$. The block "Feature Extraction, Training set generation" in Fig. 3 represents the process of extracting these features and generating a set of training samples. Using the training sample, in our framework, we train a machine learning based theoretical model, which we later use for answering queries. As sketches are mainly applied as high speed data stream scenarios, the learning model should be easy and fast to train. As a results, we mainly use simple learning regression as our learning algorithm. The block "Training" in Fig. 3 represents the process of training. Finally, 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 Fig. 3 represents the process of answering the queries.

3.2.3 Optimization

As mentioned above, existing sketches always contain two steps of optimization: 1): assuming network traffic characteristics and storing extra information of noise 2): estimating and removing noise according to extra information. These processes depend on assumptions and extra memory usage. Our training features are extracted from a normal sketch without any assumptions or extra information, so intuitively they may not contain enough information on the amount of noise in the sketch. Moreover, a lot of previous sketches focus on separate elephant flows and mice flows because noise has different influences on them. For elephant flows, previous works always focus on accurately recording their information. For mice flows, previous works focus on removing noise for them instead. In our machine learning sketch, it is also not proper to use a single model for all flows.

To solve above problems, we proposed an enhanced machine learning sketch. Its main idea is to classify flows into different clusters. We firstly classify flows into different clusters according to their basic features we extracted using a normal sketch. We will train different models for different clusters of flows. For flows that are easily influenced by noise (typically mice flows), we further add more features with noise information in our machine learning model. Extra features with noise information can help machine learning model estimate and remove noise for those flows. For flows that are difficult to be influenced by noise (typically elephant flows), we hope the machine learning model will focus on flows' own characteristics instead of noise. So they will have relatively fewer features for training.

4 Case Studies

Now, we present three case studies to show how we apply our framework to different sketches that are used to measure three well-known metrics: flow size, top-k flows, and number of flows. In presenting these case studies, we will also give concrete examples of features, the format of training samples, machine learning algorithms for training, and the process of answering queries.

4.1 Estimating Flow Sizes

4.1.1 Basic Version

In applying our framework to the sketches used 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. Therefore, our very first objective is to automatically identify the flows that can potentially experience high error.

To explore how machine learning can identify such flows, we took a real network traffic traces containing 10M packets and generated a CM sketch from it. By conducting this experiment with many different packet sets, we studied the d hashed counters of all flows whose estimates experienced high errors. 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. Based on this observation, we propose to use a simple and fast method to distinguish between these flows and the flows whose estimates experience acceptably low error. Given an incoming flow, we compute hash functions and get the two smallest counters v_1 and v_2. If $|v_1 - v_2|$ is greater than a threshold θ, where $\theta \geqslant 0$, we regard this flow as an error-prone flow. Next, we describe how we apply our machine learning framework to improve the accuracy of sketches in estimating the sizes of error-prone flows. More specifically, we describe how each block in Fig. 3 works when augmenting machine learning with prior sketches (i.e., CM and CU, CSM sketches) that are used to estimate flow sizes.

Building Regular Sketch: On arrival of each packet, the algorithm of the sketching technique under consideration uses its usual method to record information about that packet in the regular sketch. In the improved version, we also added some extra bits in each counter to stored addresses,which will be discussed later.

Sampling: Depending on the sampling rate set by the network administrator, we sample the packets for building learning sketch, and get the sampled set $\mathcal{S}$.

Building Learning Sketch: We generate a learning sketch using the packets in the sampled set $\mathcal{S}$. In case of the problem of flow size estimation, the framework also maintains a hash table to record the actual values of flow sizes, which are later used as ground truth in training machine learning models. On inserting each packet of the sampled set $\mathcal{S}$ into the learning sketch, we also insert the flow ID of this packet into the hash table which is used to accurately record the flow-sizes. One issue of building learning sketch is the determination of the sketch size. Intuitively, the sketch size should be proportional to the number of packets to provide a compatiblity of the learning sketch and regular sketch.

Feature Extraction and Training Set: We first traverse through all flow IDs in the hash table and identify the error-prone flows by comparing the smallest counter with the threshold θ. After identifying all error-prone flows, 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. For any given error-prone flow, the values of its d hashed counters in the learning sketch serve as the features. 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. In addition, linear regression is a simple machine learning model. So we do not need to worry too much about over-fitting. We also do not need to partition validation set and can apply all data to train the model.

Training: We apply linear regression on the training samples to obtain a linear regressive model that can estimate the size of any given error-prone flow using its d hashed counters.

The hypothesis function is shown as following, where α_is are the learning parameters, *flowcount* is the queried value, and $A_i[h_i(f)\%w]$ represents value of the i^{th} counter.

$$flowcount = \sum_{i=0}^{d-1} \alpha_i A_i[h_i(f)\%w]. \tag{1}$$

The equivalent form written below extracts the noise term explicitly. This form exhibits our object of learning the noise more clearly. The first term on the right is the query value of normal sketch, and the second term is the noise term. The noise term is a linear function of corresponding counter values.

$$flowcount = min\{A_i[h_i(f)\%w]\} + \sum_{i=0}^{d-1} \beta_i A_i[h_i(f)\%w]. \tag{2}$$

Querying: 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. 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.

4.1.2 Enhanced Version

Although our basic version is quite intuitive and works quite well according to our experiments, it has two drawbacks.

1) Only using d hash counters as features may not provide enough information on noise and hash conflicts. Typically, a counter in a normal CM sketch consists of target flow ID and other flow IDs due to hash conflicts. We call them *conflict IDs* in the rest of our paper. For d hashed counters, they only have our target ID in common. Consequently, when learning only from d hashed counters, our features may not provide enough information on conflict IDs.

2) We still made empirical assumptions. Our basic version relies on the recognization of error-prone flows. We find that it is quite hard to decide the threshold θ, and the performance of our algorithm is quite sensitive to this hyperparameter. Although the recognization of error-prone flows prevents applying a single model for all flows, our recognization rule is an empirical rule based on our observation.

To solve the above drawbacks, we further proposed our enhanced version. Our enhanced version adds new features on noise information (conflict IDs) for each counter with a small counter value. It also no longer differentiates error-prone flows with certain rules. Instead, the enhanced version classifies flows into different clusters and train different models for each cluster.

Enhanced Version One with New Features Selection: When a flow has been hashed to d counters in a normal CM sketch, we usually think the smallest counters value contains most of the ID's information. So, we can use the smallest counter value of a certain ID to provide its information if this ID acts as noise (conflict ID). A naive way to store these extra features is to add pointers to each counter. Specifically, for each insertion, the inserted ID is hashed into d different counters. We store the address of the smallest counter in all these d counters and refer the address stored in each counter as its pointed counter. Each counter and its pointed counter contain the same information of a certain flow ID which may be noise in the counter. In the rest of our paper, we refer this method as *Pointer-Based Version* or *Enhanced Version One*. However, the pointer will lead to heavy memory overhead, we need to find a more efficient way to store these extra features.

Enhanced Version Two with New Features Storage: Assuming we allocate b bits for each counter in the CM sketch, we divide each counter into two equal-sized parts (both $b/2$ bits). *The counter uses the second part to record the counter value and the first part to store the extra feature.* During insertion, we still try to store the smallest counter value in all these d counters' first parts. According to the value in the smallest counter and remained counters, there are two different cases.

Case 1: For values in the smallest counter and a remained counter, if at least one of them needs more than $b/2$ bits to represent, no further operation is needed and we will try to store the smallest counter value into the next remained counter.

Case 2: If both the smallest counter value and the value in a remained counter takes less than $b/2$ bits to represent, we store the smallest counter value into the first part of the remained counter. After that, we turn to the next one.

As the insertion continues, we will increment the second part of hash counters. If $b/2$ bits are no longer enough, we will clear its first part and the counter becomes a normal counter with only one part. We add an extra bit in each counter as a flag to identify if the counter has two parts or one. In summary, if the value in the counter is big, then no extra features will be stored in it and vice versa. We refer this version as *Enhanced Version Two.*

Flow Classification: For each flow, each of its d hash counters may contain two parts (according to the flag). Consequently, there are $d+1$ different cases with $d, d+1 \cdots 2d$

features respectively, thus they are automatically classified into $d+1$ clusters according to their number of features. Intuitively, if an ID has more hash counters with big values, it is more likely to be an elephant flow and noise will do less harm to it. So the more hash counters with big value an ID has, the less feature and information it will need to represent the noise. This intuition is consistent with our classifying method. In a special case, an ID only has d features will indicate the values of its d counters are all big. In such cases, a normal CM sketch may be accurate enough and the corresponding trained model for the group may be very closed to a normal CM sketch: only the parameter of the smallest counter has none-zero value 1.

Training: As flows are clustered according to their own size and the size of conflict flows in their hash counters (based on relative size with $2^{b/2}$), so flows and their noise in each cluster have similar distribution. For the j^{th} model ($j = 0 \cdots d$), there are $d + j$ features. Consequently, we will train a linear regression model for each cluster. Let α_is and α'_is be the learning parameter, *flowcount* be the queried value, and D_i represents value of the first part in the i^{th} two-part counter. Then the improved formula is:

$$flowcount = \sum_{i=0}^{d-1}(\alpha_i A_i[h_i(f)\%w]) + \sum_{i=0}^{j-1}(\alpha'_i D_i). \tag{3}$$

Specifically, all counters in our enhanced versions are sorted in order according to their values.

Querying: In the enhanced vision, we will first see how many features an ID has and selected the corresponding model according to its feature number. Then we estimate its size using those features and the model.

4.2 Estimating Top-K Flows

Unlike flow size estimation which suffers from only one type of error, i.e., the estimation error, 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, i.e., the error in the estimate of the size of flow returned by the algorithm of the sketching technique. Misclassification error is defined as the percentage of flows among the top-k flows returned by a scheme that are actually not among the top-k flows and were erroneously declared by the scheme as being among the top-k flows. The misclassification error occurs primarily due to the over-estimation error of the CM sketch in early periods, 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*.

We have two tasks in top-k flow estimation problem: 1) *classification task* to reduce the misclassification error, and 2) *estimation task* to reduce the estimation error. Before describing how we apply our machine learning framework to solve these two tasks, we first present our proposed method to reduce the misclassification error. For reducing 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. The explanation to this fact is simple. After an ID being inserted in a min-heap, its increasing size will be recorded accurately. For a mice flow inserted in the min-heap, it will be rarely incremented or it is not a mice flow. This argument is valid for both regular and learning min-heaps. Consequently, if we can keep a track of how many times the counter of each flow in the min-heap is incremented after it is inserted i.e. the final and initial values of each flow in the min-heap, we can easily identify the flow IDs in the min-heap that are mice flows. To enable such tracking, we add a third field in each node of the regular min-heap, namely initial counter. Whenever a new flow ID is inserted into the regular min-heap, the value of the initial counter field is set to the current estimate of its size, as determined by the regular CM sketch. Next, we describe how we apply our machine learning framework to reduce the misclassification and estimation errors.

Building Regular Sketch: On arrival of each packet, we add the information of that packet to the regular CM sketch and update the regular min-heap and regular hash table using the method described in Section 2.2.

Sampling: Depending on the sampling rate set by the network administrator, our framework samples the packets for building learning CM sketch and corresponding learning min-heap, and get the sampled set $\mathcal{S}$.

Building Learning Sketch: We generate a learning CM sketch and corresponding learning min-heap and learning hash table using the packets in set $\mathcal{S}$. In the improved version, different from the regular sketch, we set the size of min-heap to $2k$, so that there will be approximately same positive samples and negative samples in the min-heap and our machine learning model generalize better.

Feature Extraction and Training Set: 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 chose 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. We found that the values in the counter field in the min-heap can represent the feature of flow sizes much better than the d hashed counters of each flow in the learning CM sketch. Thus in the improved version, we only use the value in the initial counter field and the counter field in the learning min-heap as features i.e. the finial values and the times of increment for a linear model.

For *estimation task*, again, similar to the flow size 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, as shown in Equations 1 and 2, to train the linear model. To reduce the hash conflicts in the CM sketch and make a better use of memory, for a flow in the min-heap, we only increment the counter node in the min-heap while d hashed counters in CM sketch will remain the same. When flow in the min-heap has been replaced by another flow, we add its incremented times in the min-heap to d hash counters at one time.

Training: We choose to use logistic regression as our machine learning algorithm for the classification task, and use linear regression for the estimation task. For the classification task, we also tried support vector machine, but observed similar accuracy. We finally use the logistic regression model due to its very low computational and space complexity.

The hypothesis function is shown as following, where α_is and β are learning parameters.

$$label = 1/(1 + exp(-z)) \tag{4}$$

$$z = \sum_{i=0}^{d-1} \alpha_i A_i[h_i(f)\%w] + \beta(final(f) - initial(f)) \tag{5}$$

The hypothesis function takes exactly the form of logistic regression. $label$ is a boolean value representing whether a flow is in top-k or not. $label = 0$ means that the flow is not a top-k flow and vice versa. The exponent is a linear combination of chosen features which we described above under `Feature Extraction and Training Set` Part. $A_i[h_i(f)\%w]$ represents the counter value. $initial(f)$ and $final(f)$ represents the initial and final values of a flow in the heap.

Enhanced Version: In our enhanced version, similar to Section 4.1.2, we add more features and train different models for different clusters. We add an extra term $\sum_{i=0}^{j-1}(\alpha'_i D_i)$ to equation 5.

Querying: To answer a query about the top-k flows, for each flow with ID f in the regular min-heap, we estimate its probability of belonging to top-k flows by applying logistic regression model. We can simply regard the flows with a probability more than 0.5 as a top-k flow or we can set the min-heap a little larger than k and get the flows with the k highest probability as the top-k flows and use the final value in the min-heap to estimate its size. Finally, we return the flow IDs remaining in the min-heap and the corresponding estimated sizes.

4.3 Estimating Number of Flows

Accurately estimating the number of flows is one of the most important tasks in network management and monitoring [7]. 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. Next, we describe how we apply our machine learning framework to reduce the error in estimating the number of flows.

Building Regular Sketch: On arrival of each packet, we use the algorithm of FM sketch to record information about that packet in the regular sketch using the method described in Section 2.3.

Sampling: Depending on the sampling rate set by the network administrator, we sample the packets for building learning FM sketch and get the sampled set $\mathcal{S}$.

Building Learning Sketch: 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. Let the number of training samples we want to generate be z. To generate the k^{th} learning FM sketch, where $1 \leqslant k \leqslant z$, we create an FM sketch using only $\lambda_k\%$ of all flow IDs in the sampled set S, where $\lambda_k < \lambda_{k+1}$ and $\lambda_z = 100$. To select $\lambda_k\%$ of flow IDs from all flow IDs in the sampled set, for each flow ID, we decide to use it to create the kth FM sketch with probability of $\lambda_k\%$. Recall from Section 3 that if n sampled packets have a flow ID f, then that flow ID will have n entries in the sampled set $\mathcal{S}$.

Feature Extraction and Training Set: As aforementioned, one learning FM sketch can act as only one training sample. We generate multiple learning FM sketches. 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 1 in the i^{th} array) as features, as a result of extensive experiments with different 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 hight-bit is also a monotonically increasing function of the number of flows.

Training: We minimize cross-entropy between predicted values and real values with stochastic gradient descent [44] on the training samples to obtain a model that can predict the exponent part in query formula using the locations of low-bits and high-bits in the regular FM sketch.

The hypothesis function is shown as following.

$$flowsize = \gamma 2^{\frac{1}{d}\left(\sum_{i=0}^{d-1}\alpha_i L_i + \sum_{i=0}^{d-1}\beta_i H_i\right)}, \tag{6}$$

$flowsize$ represents the queried flowsize and the exponent takes exactly the form of linear regression. αs, βs, and γ are learning parameters. L_i and H_i represent the low-bit and high-bit of the i^{th} FM sketch respectively.

Querying: 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, L_i, of low-bits and another d locations, H_i, of high-bits in the given regular FM sketch and get the exponent part of query formula. Next, we use the estimation formula of FM sketch to estimate of number of flows.

5 Implementation

In this section, we describe the system that we developed to evaluate the performance improvement of various sketches due to our machine learning framework. The codes are released at Github [2]. We also justify the sampling ratio we used and the maximum rates at which our implementation can record information from packets and accurately respond to queries.

5.1 Overall System Design

As shown in Fig. 4, there are two parts in our architecture: a switch and a server. The switch is responsible for generating the *regular* sketches (that include CM, CU and FM sketches) using packets arriving at the switch from all of its interfaces and passes these sketches to the server every few seconds over its direct Ethernet connection with the server. The switch is also responsible for producing the sampled set $\mathcal{S}$.

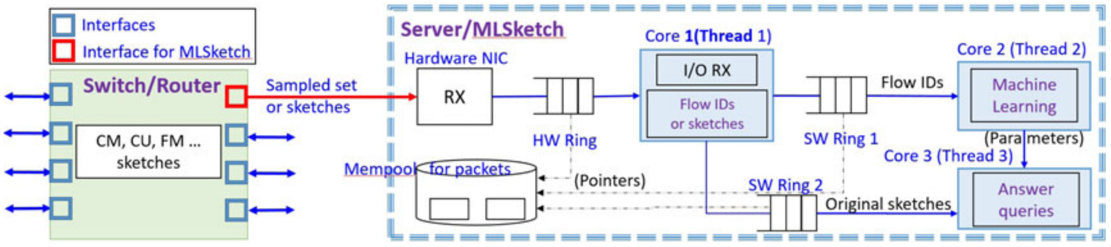

Fig. 4. Block diagram of the implementation of our machine learning framework.

The server has two network interface cards: one supports DPDK [1] and is used to receive real or synthetic network traffic, whereas the other is used to receive queries and return results.

Computer Settings: We used two servers, each 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, which emulates the packets being captured and processed at line-rate.

5.2 Implementation of Our ML Framework

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, and each thread is bound with a CPU core. To receive packets from the switch, we use DPDK [1]. Memory pool, a data structure defined and managed by DPDK, is used in the server to store packets received from NIC (i.e., payload of flow-IDs extracted from sampled packets as well as payload of regular-sketch), which can accelerates packet processing by eliminating packet copying between kernel space and user space. 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 and polls 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.

5.3 Sampling Rate and Practicality

Next, through some basic calculations, we show that our experimental setup can easily perform the three tasks of estimating the flow-size, estimating the top-k flows, and estimating the number of flows on traffic moving at 40Gbps*100 interfaces. The following discussion should also alleviate concerns about the practical usability of employing machine learning based techniques to improve the accuracy of sketches.

From our very comprehensive set of experiments, we observed that the accuracy of sketches decreases very slowly when we decrease the sampling ratio from 1 in 5 packets to 1 in 1000 packets and we will discuss in more detail soon. Consequently, a sampling ratio of 1 packet every 100 packets is a reasonable choice and still enables us to improve the accuracy of existing sketches significantly. As our NIC and the supporting architecture in the server can process data at the full line rate, when sampling at a rate of 1 in 100 packets, the hardware of our system can essentially process traffic seen by all ports of a 100-port switch. Furthermore, the switch only sends the flows ID of the sampled packets to the server and not the entire packets. Suppose the size of the flow ID is one-third of the size of the entire packet (usually the packets are much larger than $3 \times$ flow ID size), then the hardware of our server can essentially process traffic seen by up to $3 \times 100 = 300$ switch interfaces. Furthermore, as shown in Fig. 1 of [42], the links are often over-provisioned and the link utilizations often lies below 20%. Consequently, hardware of our server can essentially process traffic seen by up to $300/20\% = 1500$ switch interfaces. This is a large enough number for any commercial switch and router.

From our experiments, we also observed that the machine learning model generated by thread 2 is valid for a reasonably long amount of time (see Fig. 8). Thus, thread 2 can generate machine learning models periodically and not continuously. According to our experimental results (see Fig. 7), when the sampling rate is $1/100$, thread 2 takes about 0.165 seconds to process 10M packets, using a single CPU core. This implies that our machine learning implementation

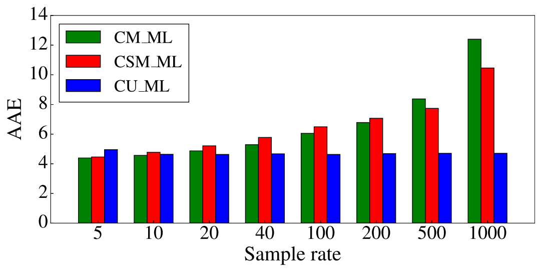

Fig. 5. Absolute error versus sampling rate.

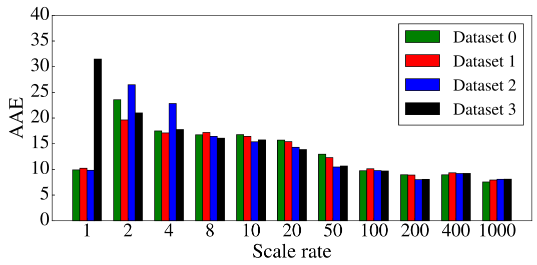

Fig. 6. Average absolute error versus scale rate.

can easily keep up with a line rate of $10\text{M} \times 84 \times 8/0.165 \approx 40\text{Gbps}$, assuming the minimum size of each Ethernet packet is 84 bytes. When average link utilization is less than 10% [42], using a single CPU, the software of our system can easily support a 40 Gbps switch with 10 interfaces. On a 16 core CPU, while utilizing only 10 cores for thread 2 and remaining cores for threads 1 and 3, the software of our system can easily support traffic seen by a 40 Gbps switch with 100 interfaces using the strategy of training always. Furthermore, according to Fig. 8, using training per hour instead of training once, many more interfaces can be supported at the cost of a little loss of accuracy.

6 Experimental Results

In this section, we present the results from our experiments for each of the three case studies that we presented in the paper. Although we have improved version of algorithms for flow sizes and top-k flows estimation tasks, they only shows differences with the basic version in Average Absolute Error and Average Relative Error. So for experiments used to determine hyper parameters, we only use the basic version. The codes of sketches are released on Github [2].

Traffic Traces: We collected real network traffic traces from a tier-1 router. We captured 10 minutes of network traffic each hour on two days. The number of packets in each 10 minute traffic trace ranged from 876303 to 1124480, with 41.81% flows having less than one packet, while some flows also had more than 30000 packets. We identify flows using the standard 5-tuple. We observed that the average flow size for different 10-minute traces ranged from 8.67 to 10.04 packets per flow while the standard deviation ranged from 102.7 to 146.9.

6.1 Evaluation for Flow Size Estimation

In this section, we use suffixe _ML to represent that a sketch uses machine learning optimization and use suffixe _EML1/_EML2 to represent the pointer-based versions and enhanced version respectively.

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.

AAE is defined as:

$$AAE = \frac{1}{n}\sum_{i=1}^{n} |\hat{r}_i - r_i|$$

ARE is defined as:

$$ARE = \frac{1}{n}\sum_{i=1}^{n} \frac{|\hat{r}_i - r_i|}{r_i},$$

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: We have three parameters to select: d, w, and sampling rate. In all our experiments, we allocate a fixed amount of 1 Mbit memory to all sketches. Allocating the fixed amount of memory allows us to do a fair comparison. We also observed from our experiments that using a value of $d = 3$ results in high accuracy. Therefore, we chose to use $d = 3$ in all our experiments. The size of each counter for sketches is 32 bits.

Using these value of d and w, we measured the AAE achieved by the three sketching techniques for flow size estimation (i.e., CM, CU, and CSM) augmented with machine learning with sampling rates varying from 1 in 5 packets to 1 in 1000 packets. Fig. 5 plots the AAE achieved by each of these three techniques. It shows that the improvement in AAE 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 for training machine learning models.

Learning Sketch Size: Now, we determine the appropriate size of learning sketch. The scale rate is defined as the size of original sketch devided by the size of learning sketch. In this experiment, We use 1/1000 as our sample rate to achieve a wide range of scale rate changes. We can vary our scale rate from 1 to 1/1000 to better present the relation between sample rate and corresponding scale rate. The intuition is that scale rate should be inversely proportional to sample rate. Experimental results further verify the hypothesis. Fig. 6 illustrates that with a sample rate of 1/1000, a learning sketch with scale rate of 1000 has the best performance in terms of accuracy.

Training Frequency: Next, we determine the frequency at which one should retrain the machine learning model for optimum improvement in accuracy. For these experiments, we use packets from our 10-minute traces that we collected every hour on the first day. From each 10-minute trace, we use the first 10M packets. To identify optimum training frequency, we performed three sets of experiments. In the first set of experiments, we calculated AAE for the regular CM sketch. In the second set of experiments, we calculated AAE using the first trace, and built a machine learning model for each dataset (10M packets). Fig. 8 plots the AAE ratio of these two sets of experiments, labeled “training always”. In the third set of experiments, we calculated AAE using the first dataset (10M packets) of the first trace with machine learning, and use the trained model to predict the first 23

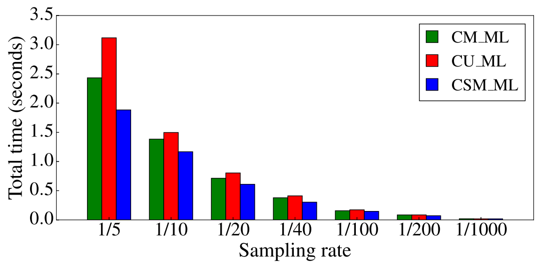

Fig. 7. The time overhead of machine learning using different sampling rate.

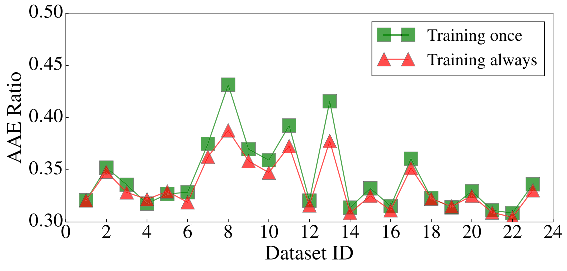

Fig. 8. Training once versus. training always.

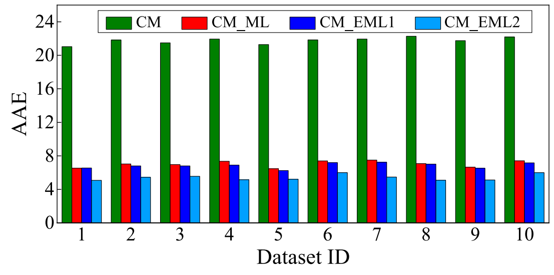

Fig. 9. AAE of CM sketches for different datasets.

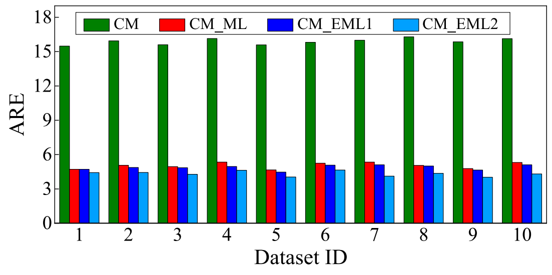

Fig. 10. ARE of CM sketches for different datasets.

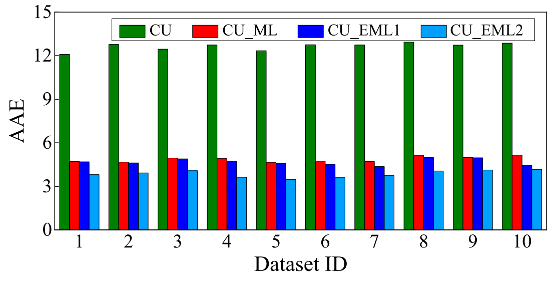

Fig. 11. AAE of CU sketches for different datasets.

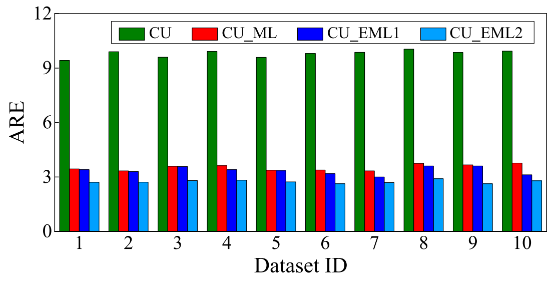

Fig. 12. ARE of CU sketches for different datasets.

datasets of the second trace. Fig. 8 plots the AAE ratio calculated from the third and first set of experiments, labeled "training once". It shows that training on a decent amount of data collected once an hour 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.

Next, we present the results from our experiments in terms of the evaluation metrics defined earlier. The results that we present for each sketch in Sections 6.1.1 and 6.1.2 next are generated by emulating the traversal of each of the 10-minute traffic traces from the switch machine and generating that sketch with or without machine learning.

6.1.1 Evaluation With CM Sketch

Our experimental results (Figs. 9 and 10) show that the AAE of CM sketch with machine learning typically reduces to about one third of the normal CM sketch's AAE, from about 20 on average to about 6 on average. The ARE of regular CM sketch is above 15 on average among different datasets. That is because that most of the flows are mice flows, leading to high ARE. From our experimental results, shown in Fig. 10, the ARE also decreased from 15 on average to about 4, with ARE ratio of about 0.25. We observe from this figure that CM_ML, CM_EML1 and CM_EML2 achieved a smaller error rate compared to the regular CM sketch. The performances of CM_ML and CM_EML1 do not show much difference due to the overhead of pointers. CM_EML2 showed about 20% performance compared to CM_ML.

6.1.2 Evaluation With CU Sketch

Our experimental results (Figs. 11 and 12) show that the AAE of CU sketch with machine learning typically reduces to above one third of the normal CU sketch's AAE, from about 12 on average to about 5 on average. The ARE of regular CU sketch is above 15 on average among different datasets. From our experimental results, shown in Fig. 12, the ARE also decrease from 10 onn average to about 3, with ARE ratio of about 0.33.

We observe from this figure that CU_ML, CU_EML1 and CU_EML2 achieved a smaller error rate compared to the regular CU sketch. CU_EML2 also showed better performance compared to CU_ML and CU_EML1. We may find that the it has similar amount of improvement (about 20%) compared to CM_ML and CM_EML even though for CU sketches, counters share much less same noise.

6.1.3 Evaluation with CSM Sketch

Our experimental results (Figs. 13 and 14) show that the AAE of CU sketch with machine learning typically reduces

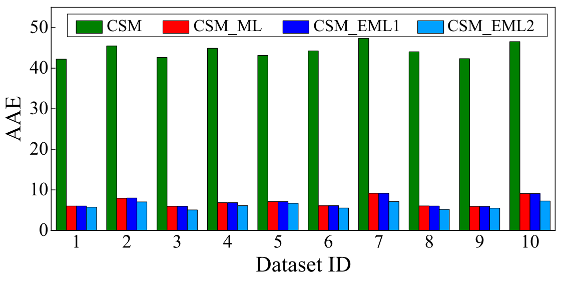

Fig. 13. AAE of CSM sketches for different datasets.

to above one seventh of the normal CSM sketch's AAE, from about 45 on average to about 6.5 on average. The ARE of regular CSM sketch is above 30 on average among different datasets. From our experimental results, shown in Fig. 14, the ARE also decreased from 30 on average to about 5, with ARE ratio of about 0.16.

We observe from this figure that CSM_ML, CSM_EML1 and CSM_EML2 achieved a smaller error rate than regular CSM sketch. CSM_EML2 showed better performance compared to others.

6.1.4 Training Overhead Versus Sampling Rate

In our implementation, the time to generate learning sketch, generate training samples, and train linear regression model from 10M packets decreased from an average of 3.201s to 0.108s when decreasing the sampling rate from 1 in 5 to 1 in 1000 packets. Fig. 7 plots the total time for different sketching techniques to generate learning sketch, generate training samples, and train linear regression model from 10M packets. For the sampling rate of 1 in 100 packets, this time is just 0.165 sec. Even if we train a model once every hour, 0.165 seconds are negligible in one hour. Thus, machine learning can be easily incorporated in real switches and routers while keeping up with the line rate.

6.2 Evaluation for Top-K Flow Estimation

In this section, we use CM_Heap to represent a CM sketch and a min-heap that are used to estimate sizes of top-k flows, and CM_Heap_ML to represent that we are using machine learning to improve the accuracy. We also use CM_Heap_EML1 to represent the pointer-based version and CM_Heap_EML2 to repersent the enhanced version. We use the same two metrics, AAE and ARE, to quantify the performance of flow size estimation accuracy of sketches for top-k flow estimation. To evaluate these metrics, we calculate the difference between the real size of a top-k flow and its size recorded in the heap. For a top-k flow that hasn't been recalled, we treat its measured size as 0.

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 9 to 29. Figs. 15 and 16 plot the AREs and AAEs, respectively, of CM_Heap and CM_Heap_ML, CM_Heap_EML1, as well as CM_Heap_EML2. We can find that the CM_Heap with machine learning shows significant improvement compare with a normal CM_Heap. Besides, the pointer-based version and enhanced version performed much better than CM_Heap_ML, for it uses linear regression to fix the value when a flow is inserted into the heap. We find that in top-k task, our machine learning sketches shows less improvement compared with flow size estimation task. The main reason behind this is that top-k flows are big in size and are difficult to be influenced by noise. But our enhanced machine learning sketches still show 50% improvement compared with normal sketches.

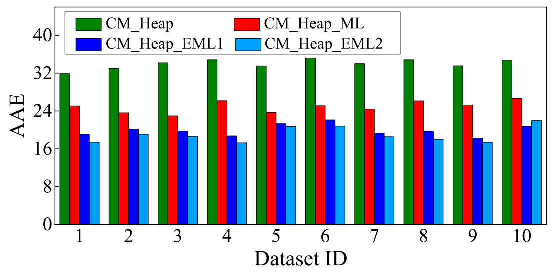

Fig. 15. AAE for the top-k flows on different datasets.

6.3 Evaluation for Flow Number Estimation

In this section, whenever we talk about FM sketch that uses machine learning, we represent it with FM_ML sketch. The evaluation metrics that we use in this section are relative error (RE) and ARE. In our implementation, we used $d = 20$ and $w = 32$. For the set of experiments presented in this section, the memory used by the sketch is, therefore, only 80 bytes, which is small enough to be stored even inside the CPU registers.

Our experimental results show that in traces for which REs of FM sketch are larger than 30%, the 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. Fig. 17 plots the REs of FM and FM_ML sketch in 10 randomly chosen such 10M packet traces for which RE of FM sketch is larger

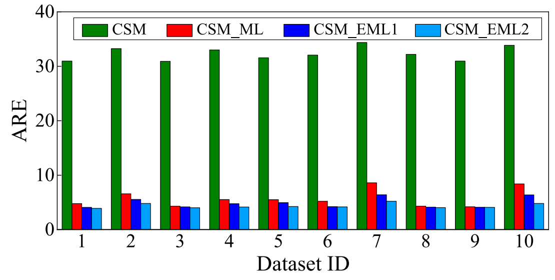

Fig. 14. ARE of CSM sketches for different datasets.

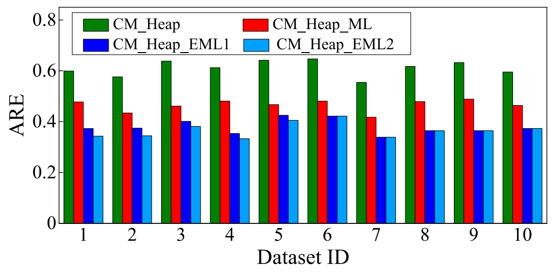

Fig. 16. ARE for the top-k flows on different datasets.

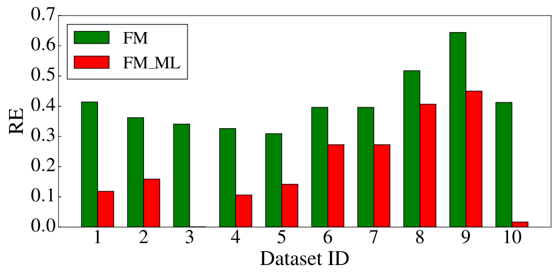

Fig. 17. RE comparison (RE of FM sketch is $> 30\%$).

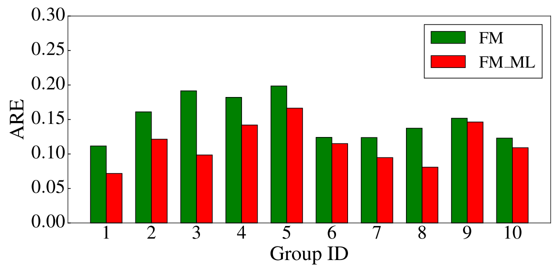

Fig. 18. ARE comparison (RE of FM sketch $< 30\%$).

than 30%. We clearly observe from this figure that machine learning significantly reduces the RE of the FM sketch.

Our experimental results also show that in traces for which RE of the FM sketch is smaller than 30%, the AREs of the FM_ML sketch are up to 49% smaller than the AREs of FM sketch with a mean of 23%. Fig. 18 plots the AREs of FM and FM_ML sketch in 10 groups of datasets, where each group contains 10 randomly chosen 10M packets, for each of which, the RE of FM sketch is smaller than 30%. We conclude from Figs. 17 and 18 that when the the FM sketch already has reasonably small error, then machine learning reduces the error only nominally, but when FM sketch has large error, then machine learning helps in reducing the error remarkably.

7 Conclusion

Due to the urgent requirement of accurate estimation of network traffic, the sketching techniques have drawn significant attention in recent years. In this paper, we have proposed a *generic 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 three typical flow-level metrics (flow sizes, top-k flows, and number of flows) and apply our framework to them to demonstrate the effectiveness of our framework.

Our machine learning framework can be applied to not only most of the existing sketches but also to other probabilistic data structures. We hope that this work would spark more research in the area of automating the sketching techniques using machine learning.

References

[1] DPDK websit, [Online]. Available: http://dpdk.org/
[2] “Experimental and implemention codes,” [Online]. Available: https://github.com/spartazhihu/ML-Sketch
[3] N. Bandi, A. Metwally, D. Agrawal, and A. El Abbadi, “Fast data stream algorithms using associative memories,” in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2007, pp. 247–256.
[4] Z. Bar-Yossef, T. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan, “Counting distinct elements in a data stream,” *Randomization and Approximation Techniques in Computer Science*. Berlin, Germany: Springer, pp. 1–10, 2002.
[5] G. Bianchi, K. Duffy, D. Leith, and V. Shneer, “Modeling conservative updates in multi-hash approximate count sketches,” in *Proc. Int. Teletraffic Congr.*, 2012, pp. 1–8.
[6] M. Charikar, K. Chen, and M. Farach-Colton, “Finding frequent items in data streams,” in *Automata, Languages and Programming*, Berlin, Germany: Springer, 2002.
[7] G. Cormode, “Sketch techniques for approximate query processing,” in *Foundations Trends in Databases*, Boston, MA, USA: NOW Publishers, 2011.
[8] G. Cormode and M. Garofalakis, “Sketching streams through the net: Distributed approximate query tracking,” in *Proc. 31st Int. Conf. Very Large Data Bases*, 2005, pp. 13–24.
[9] G. Cormode and S. Muthukrishnan, “An improved data stream summary: The count-min sketch and its applications,” *J. Algorithms*, vol. 55, vol. 1, pp. 58–75, 2005.
[10] G. Cormode and S. Muthukrishnan, “Summarizing and mining skewed data streams,” in *Proc. SIAM Data Mining Conf.*, 2005.
[11] G. Cormode and S. Muthukrishnan, “What’s hot and what’s not: Tracking most frequent items dynamically,” *ACM Trans. Database Syst.*, vol. 30, vol. 1, pp. 249–278, 2005.
[12] G. Cormode and S. Muthukrishnan, “What’s new: Finding significant differences in network data streams,” *IEEE/ACM Trans. Netw.*, vol. 13, vol. 6, pp. 1219–1232, 2005.
[13] F. Deng and D. Rafiei, “New estimation algorithms for streaming data: Count-min can do more”, 2007.
[14] M. Durand and P. Flajolet, “Loglog counting of large cardinalities,” in *Proc. Eur. Symp. Algorithms*, 2003, pp. 605–617.
[15] C. Estan and G. Varghese, “New directions in traffic measurement and accounting,” *Comput. Commun. Rev.*, vol. 32, no. 4, 2002.
[16] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, “Summary cache: A scalable wide-area web cache sharing protocol,” in *Proc. ACM SIGCOMM Conf.*, 1998.
[17] P. Flajolet, “On adaptive sampling,” *Computing*, vol. 43, vol. 4, pp. 391–400, 1990.
[18] P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier, “HyperLogLog: The analysis of a near-optimal cardinality estimation algorithm,” *AofA: Anal. Algorithms*, pp. 137–156, 2007.
[19] P. Flajolet and G. N. Martin, “Probabilistic counting algorithms for data base applications,” *J. Comput. Syst. Sci.*, vol. 31, pp. 182–209, 1985.
[20] P. B. Gibbons, “Distinct sampling for highly-accurate answers to distinct values queries and event reports,” *Proc. VLDB*, vol. 1, pp. 541–550, 2001.
[21] K. Hu, H. K. Chandrikakutty, Z. Goodman, R. Tessier, and T. Wolf, “Dynamic hardware monitors for network processor protection,” *IEEE Trans. Comput.*, vol. 65, vol. 3, pp. 860–872, Mar. 2016.
[22] Q. Huang *et al.*, “Sketchvisor: Robust network measurement for software packet processing,” in *Proc. Conf. ACM Special Int. Group Data Commun.*, 2017, pp. 113–126.
[23] Q. Huang and P. P. Lee, “LD-sketch: A distributed sketching design for accurate and scalable anomaly detection in network data streams,” in *Proc IEEE Conf. Comput. Commun.*, 2014, pp. 1420–1428.
[24] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen, “Sketch-based change detection: Methods, evaluation, and applications,” in *Proc. 3rd ACM SIGCOMM Conf. Internet Meas.*, 2003, pp. 234–247.
[25] B. Lahiri, J. Chandrashekar, and S. Tirthapura, “Space-efficient tracking of persistent items in a massive data stream,” in *Proc. 5th ACM Int. Conf. Distrib. Event-Based Syst.*, 2011, pp. 255–266.
[26] Y. Li, R. Miao, C. Kim, and M. Yu, “FlowRadar: A better netflow for data centers,” in *Proc. 13th USENIX Conf. Netw. Syst. Des. Implementation*, 2016, pp. 311–324.
[27] Z. Liu *et al.*, “One sketch to rule them all: Rethinking network flow monitoring with univmon,” in *Proc. ACM SIGCOMM Conf.*, 2016, pp. 101–114.
[28] Z. Liu, G. Vorsanger, V. Braverman, and V. Sekar, “Enabling a “RISC” approach for software-defined monitoring using universal streaming,” in *Proc. 14th ACM Workshop Hot Top. Netw.*, 2015, Art. no. 21.
[29] Y. Lu *et al.*, “Counter braids: A novel counter architecture for per-flow measurement,” in *Proc. ACM SIGMETRICS Conf.*, 2008.
[30] C. Monsanto *et al.*, “Composing software defined networks,” in *Proc. 10th USENIX Conf. Netw.ed Syst. Des. Implementation*, 2013, pp. 1–13.

[31] M. Moshref, M. Yu, R. Govindan, and A. Vahdat, "SCREAM: Sketch resource allocation for software-defined measurement," in *Proc. 11th ACM Conf. Emerg. Netw. Experiments Technol.*, 2015, Art. no. 14.
[32] M. Moshref, M. Yu, R. Govindan, and A. Vahdat, "Trumpet: Timely and precise triggers in data centers," in *Proc. ACM SIGCOMM Conf.*, 2016, pp. 129–143.
[33] P. Roy, A. Khan, and G. Alonso, "Augmented sketch: Faster and more accurate stream processing," in *Proc. Int. Conf. Manage. Data*, 2016, pp. 1449–1463.
[34] J. Sanjuàs-Cuxart, P. Barlet-Ros, N. Duffield, and R. R. Kompella, "Sketching the delay: Tracking temporally uncorrelated flow-level latencies," in *Proc. ACM SIGCOMM Internet Meas. Conf.*, 2011, pp. 483–498.
[35] V. Sekar, M. K. Reiter, W. Willinger, H. Zhang, R. R. Kompella, and D. G. Andersen, "CSAMP: A system for network-wide flow monitoring," in *Proc. USENIX Symp. Netw. Syst. Des. Implementation*, 2008, pp. 233–246.
[36] M. Shahzad and A. X. Liu, "Noise can help: Accurate and efficient per-flow latency measurement without packet probing and time stamping," in *Proc. ACM SIGMETRICS Conf.*, 2014.
[37] U. Srivastava and J. Widom, "Flexible time management in data stream systems," in *Proc. 23rd ACM SIGMOD-SIGACT-SIGART Symp. Princ. Database Syst.*, 2004, pp. 263–274.
[38] M. Wang, B. Li, and Z. Li, "sFlow: Towards resource-efficient and agile service federation in service overlay networks," in *Proc IEEE Int. Conf. Distrib. Comput. Syst.*, 2004, pp. 628–635.
[39] T. Yang *et al.*, "Elastic sketch: Adaptive and fast network-wide measurements," in *Proc. Conf. ACM Special Int. Group Data Commun.*, 2018, pp. 561–575.
[40] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with opensketch," in *Proc. USENIX Symp. Netw. Syst. Des. Implementation*, 2013, pp. 29–42.
[41] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with opensketch," in *Proc. USENIX Conf. Netw. Syst. Des. Implementation*, 2013, pp. 29–42.
[42] M. Zhang, C. Yi, B. Liu, and B. Zhang, "GreenTE: Power-aware traffic engineering," in *Proc IEEE Int. Conf. Netw. Protoc.*, 2010, pp. 21–30.
[43] N. Zhang, R. Bettati, W. Yu, W. Zhao, and X. Fu, "Localization attacks to internet threat monitors: Modeling and countermeasures," *IEEE Trans. Comput.*, vol. 9, no. 12, pp. 1655–1668, Dec. 2010.
[44] T. Zhang, "Solving large scale linear prediction problems using stochastic gradient descent algorithms," in *Proc. ACM Int. Conf. Mach. Learn.*, 2004.
[45] Y. Zhang *et al.*, "CocoSketch: High-performance sketch-based measurement over arbitrary partial key query," in *Proc. ACM SIGCOMM 2021 Conf.*, 2021, pp. 207–222.

Hengrui Wang is currently working toward the bachelor's degree with Peking University, advised by T.Yang. His research interests mainly include focus on network Big Data, and data mining.

Huiping Lin is currently working toward the bachelor's degree with Peking University, advised by T. Yang. Her research interests include network measurement, Big Data, and machine learning. She published papers in JSAC and IFIP.

Zheng Zhong is currently working toward the bachelor's degree with Peking University, advised by T.Yang. His research interests include network measurement, data mining, and data stream process.

Tong Yang received the PhD degree in computer science from Tsinghua University, in 2013. He visited Institute of Computing Technology, Chinese Academy of Sciences (CAS). Currently, he is an associate professor with Computer Science Department, Peking University. His research interests include network measurements, sketches, IP lookups, Bloom filters, sketches and KV stores. He published papers in SIGCOMM, SIGKDD, SIGMOD, SIGCOMM CCR, VLDB, ATC, ToN, ICDE, INFOCOM, *etc.*

Muhammad Shahzad received the PhD degree in computer science from Michigan State University, in 2015. He is currently an assistant professor with the Department of Computer Science, North Carolina State University, Raleigh, NC. His research interests include design, analysis, measurement, and modeling of networking and security systems. He received the 2015 Outstanding Graduate Student Award, the 2015 Fitch Beach Award, and the 2012 Outstanding Student Leader Award at Michigan State University.