# DeepFlow: A Deep Learning Framework For Software-defined Measurement

Aggelos Lazaris, and Viktor K. Prasanna
Ming Hsieh Department of Electrical Engineering
University of Southern California
Los Angeles, CA
{alazaris,prasanna}@usc.edu

## ABSTRACT

Providing fine grained traffic measurement is crucial for many network management tasks such as traffic engineering, anomaly detection, traffic accounting, and load balancing. Software-defined networks can potentially enable fine-grained measurement by providing statistics for each forwarding rule of an OpenFlow-enabled switch. However, providing fine grained traffic measurement in a scalable fashion in hardware switches poses significant challenges due to the limitations in the size of the TCAMs that fit only a relatively small number of rules compared to the number of active flows in the network. In this paper, we present DeepFlow, a framework for scalable software-defined measurement that relies on an efficient mechanism that a) adaptively detect the most active source and destination prefixes in the network, b) collects fine-grained flow-size measurements for the most active prefixes and coarse grained for the less active ones, and c) uses historical measurements in order to train a cloud-based Deep Learning model that can be used to provide short-term predictions whenever exact flow counters cannot be placed at a switch due to its limited resources. Thus the number of fine-grained flows measured can increase significantly without the need to use other flow sampling solutions that loose accuracy. An extensive experimental evaluation using a prototype implementation and real network traces shows that DeepFlow can provide very high accuracy for estimating flow sizes at various aggregation levels.

## CCS CONCEPTS

• **Networks → Network performance modeling**; **Network measurement**; **Network monitoring**;

## KEYWORDS

Software Defined Networks (SDN), Traffic Measurement, Deep Learning, Long Short-Term Memory (LSTM), Traffic Predictions, Flow Size Modeling

**ACM Reference Format:**
Aggelos Lazaris, and Viktor K. Prasanna. 2017. DeepFlow: A Deep Learning Framework For Software-defined Measurement. In *Proceedings of CAN'17: Cloud-Assisted Networking Workshop (CAN'17).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3155921.3155922

## 1 INTRODUCTION

The availability of fine grained network traffic flow measurement is necessary in order to provide the information required for a large variety of tasks such as traffic engineering, anomaly detection, network accounting, network analytics, load balancing, Traffic Matrix (TM) estimation, and other network management and optimization tasks [3–8]. OpenFlow-enabled switches provide counters for each forwarding rule such as the total bytes or packets transferred. This can enable a wide variety of fine grained measurement tasks that can bring visibility in the network resources as long as the forwarding rules are not too broad to match multiple flows. However, in practice, the TCAM memory of the switches is fundamentally limited in size due to its high cost and power consumption. This causes the vast majority of SDN hardware vendors to limit the TCAMs to less than 4K L2/L3 rules, which is much smaller compared to the tens of thousands of flows that an SDN-enabled switch can concurrently forward [10], [15]. Thus, the need for efficient mechanisms that can enable more fine-grained measurement tasks despite the limited TCAM resources is mandatory in hardware switches in order to achieve a detailed view of the network.

Prior work on SDN traffic measurement has either assumed specialized hardware support (e.g. sketches) at the switches [12–14], or it has focused on specific measurement tasks only such as heavy hitter detection, or anomaly detection [9]. Moreover, in the context of TM estimation for SDN, prior work has either assumed that TCAMs have enough capacity to fit all the monitoring rules for all the flows ([15, 18]) which is not the case in practice, or when not, then the top K (K being limited by the total TCAM space available) most important flows are measured with exact match rules and the rest are measured in aggregate (e.g. [16, 17]). However, such mechanisms cannot be used to provide a global detailed view of the network.

**Contributions:** In this paper, we present a framework for prediction-assisted fine grained software defined measurement called DeepFlow, that can be deployed directly to production hardware switches. DeepFlow operates at the control layer and leverages four main observations. First, a given flow (or group of flows) as defined by a source and destination IP prefix can be measured in any of the switches that they traverse throughout the network, and its exact measurement location can be optimized such that the total number of flows measured is maximized. Second, when network flows are measured in aggregate over a period of few seconds or more, then the short-term variation that individual flows might exhibit in small time scales is averaged out, and thus the aggregated flow can be accurately modeled by an efficient time series prediction model. Third, if we periodically use flow size predictions for some of the flows in the network, then we can free up TCAM space and let the controller measure other unexplored regions of the IP space, thus resulting to more in-depth view of the network. Fourth, if the traffic dynamics in the network changes significantly at a given epoch and that portion of the IP space is not measured with exact flow measurements during that epoch, then we can detect the change

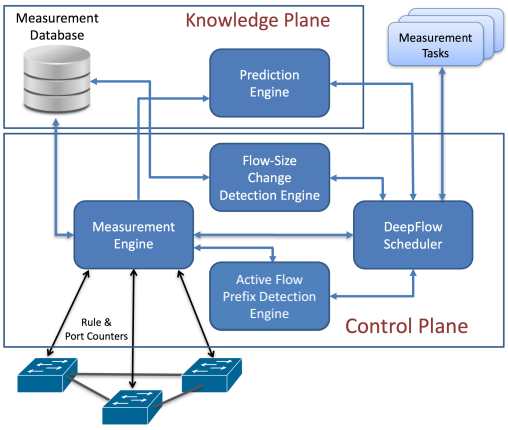


**Figure 1: DeepFlow architecture components.**

and install more fine grained measurement rules by correlating the switch port statistics data (which available "for free" in each epoch) with the routing information such that we detect with high probability which flow prefixes caused the traffic change. This way, more fine grained measurement data can be collected and stored in the cloud in order to build a knowledge plane [1] on top of the controller which can be used to train a Long-Short-Term-Memory (LSTM) recurrent neural network (deep in time) [2] and predict future flow sizes based on past measurements.

## 2 DEEPFLOW DESIGN OVERVIEW

In this section, we present the main components of DeepFlow, as shown in Fig. 1. In the analysis presented below, a flow is defined as the traffic between a source and destination IP address. However, the same concepts can be applied to higher dimensions (e.g. 5-tuples).

**Measurement Tasks**: In order to start collecting DeepFlow measurements, the network operator needs to specify the following input parameters: 1) source and destination IP prefixes of interest (if we want to monitor the whole network, DeepFlow automatically calculates the union of the source and destination IP prefixes covered by the rules in all the TCAMs and uses this as an input (e.g. `(54/8, 64/8)`), 2) the maximum flow granularity for the source and destination prefixes, expressed by a subnet mask, e.g. `(/24, /24)`, and 3) the minimum threshold $\theta$ above which the traffic of a given prefix is considered significant (e.g. `1 Mbps`). If no threshold is provided, then all the flow sizes up to the prefix granularity specified above will be considered. The threshold $\theta$ can be also defined in terms of the total available bandwidth in the switches, e.g. 0.001% of the total bandwidth.

**Active Flow Prefix Detection Engine**: One the of main challenges that DeepFlow has to overcome is the fact that the active flows in the network are not known to the controller a-priori, as well as their relative importance. The reason is that due to the limited size of the TCAM, wild-card rules are often used that can potentially match many flows (e.g. based on the destination prefix) and thus the traffic statistics of these flows cannot reveal to which exactly fine-grained flows they belong to. For this, DeepFlow uses the Active Flow Prefix Detection Engine that operates on the 2D IP space and iteratively detects what are the most active prefixes in the network.

**Prediction Engine & Measurement Database**: DeepFlow's prediction engine uses historical data stored in the cloud in a measurement database (a retention policy can be used that depends on the application needs) in order to train a deep neural network (deep in time) that is then used to generate predictions about the traffic of a given flow prefix. The prediction engine is exposed through an API to the DeepFlow scheduler. When a new prediction is needed, the DeepFlow scheduler makes a call to the API by providing the source and destination IP *prefixes*. Then the prediction engine pulls the historical data from the measurement database and generates a prediction for the next epoch. It is important to note here that it is not necessary to maintain a separate model for each flow pair, since as we will see in the subsequent sections, network flows exhibit similar traffic patterns that can be grouped together by subnet size, application, time of the day, etc.

**Flow-Size Change Detection Engine** In order to guarantee that the prediction engine does not produce large estimation errors in cases of sudden traffic changes in the network (e.g. a DDoS attack, or other traffic anomalies), DeepFlow constantly monitors the aggregated volume of all the links and uses the Flow-Size Change Detection Engine to detect which prefixes are responsible for the volume change. Then, the prefixes detected will be monitored in the next epoch with new measurement rules, instead of using predictions.

**DeepFlow Scheduler & Measurement Engine**: The DeepFlow Scheduler, given a set of measurement tasks provided by the network administrators, decides where and when to install exact flow measurements, as well as for which flows to use model predictions. For this, the scheduler uses the Measurement Engine that takes care of adding or deleting flow counter rules, as well as retrieving flow measurements from the TCAMs. For the rest of the flows, DeepFlow scheduler uses the prediction engine that predicts the flow sizes for the next epoch.

In the following sections, we formulate DeepFlow, and discuss the optimization steps that it takes in order to maximize the number of flows monitored.

## 3 PREDICTION-ASSISTED MEASUREMENT

DeepFlow employs the following two algorithms: a) the Active Flow Prefix Detection Algorithm (AFPDA) which takes care of detecting active flow prefixes, and b) the Prediction Assisted Measurement Algorithm (PAMA) that takes care of collecting measurements for the detected active flows, and interleave them with predictions in order to improve the granularity of the measurement process.

**Active Flow Prefix Detection Algorithm (AFPDA):** Unlike previous works that assume that the flows are known to the controller ([17], [16]) or they are provided as an input with the measurement task (e.g. [9]), DeepFlow operates in a flow-agnostic setting, and proactively detects the largest flows in the two-dimensional IP space by running the Active Flow Prefix Detection Algorithm (AFPDA), which is described below. When AFPDA starts, it retrieves all the forwarding rules from all the switches, as well as information regarding the ingress switches in the network (provided as an input by the network administrator during setup). The next step of

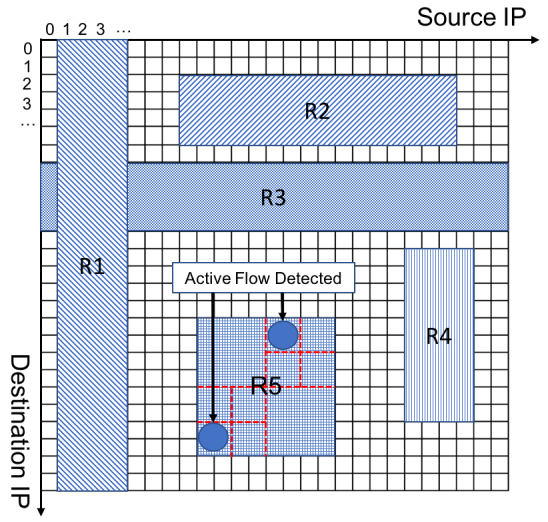


**Figure 2: An example of the set of rules at a switch, represented as shaded areas in a two-dimensional plane. The red (dashed) lines in R5 correspond to the rule splitting AFDPA performs in order to detect the two active flows in the area of R5 (i.e. the two circles).**

AFDPA is to start the flow zooming process where it tries to locate source/destination IP prefixes that have large volumes and need to be further split into longer active prefixes that can be measured separately. For this, AFDPA installs TCAM rules that have higher priority and differ only in the length of the source and destination IP prefixes (i.e. they are essentially subsets of them) compared to the initial rule, while keeping the rest of the rule values the same. This way, part of the rule's matching traffic will be offloaded to the new sub-rule, thus allowing more fine grained measurement. This way the source/destination prefix area is split into smaller regions iteratively until further splitting measures flows with smaller volume than the minimum threshold *theta*. If no threshold $\theta$ has been specified, the process will stop when the maximum prefix length for the source and destination prefixes has been reached. This process is illustrated in the example of Fig. 2 where the rule space for rule R5 is split twice until two flows (the two circles) where detected that exceeded the threshold $\theta$ with a given maximum zoom-level of `/32` in this case.

In order to find the optimal location that a monitoring rule should be installed while maximizing the total number of flows monitored (once the important flows have been determined), we proceed to formulate the problem as an Integer Linear Program (ILP) that DeepFlow solves before installing the monitoring rules for a new measurement period.

Let $\mathcal{S} = \{s_1, s_2, \ldots, s_N\}$ be the set of all switches in the network and $\mathcal{F} = \{f_1, f_2, \ldots, f_K\}$ be the set of all the (aggregated) flows that we are interested to monitor during a given measurement period. Also let $x_{i,j}$ be an auxiliary variable that takes the value 1 if flow $f_i$ is monitored with a rule at switch $j$, otherwise it is zero, and $m_j$ be the total available memory at switch $j$. Then, the problem of optimal rule placement can be written as:

$$\text{minimize} \quad \sum_{j=1}^{N} \left( m_j - \sum_{i=1}^{K} x_{i,j} \right) \tag{1}$$

$$\text{subject to} \quad \sum_{i=1}^{K} x_{i,j} \leq \cdot m_j, \ j = 1, \ldots, N \tag{2}$$

$$\sum_{i=1}^{N} x_{i,j} \leq 1, \ j = 1, \ldots, K \tag{3}$$

$$x_{i,j} \in \{0, 1\}, \ i = 1, \ldots, N; \ j = 1, \ldots, K \tag{4}$$

$$x_{i,j} \leq r_{i,j}, \ i = 1, \ldots, N; \ j = 1, \ldots, K \tag{5}$$

In the above linear program, Eq.1 maximizes the total number of rules monitored. Eq.2 guarantees that we will not try to install more rules to a switch than its available memory. Eq.3 forces the number of rules per flow to be equal to one (i.e. no flow will be monitored in two switches). Eq.4 makes sure that the auxiliary variable $x_{i,j}$ is binary, and Eq. 5 that the rules are installed in some of the switches in the path of each flow where $r_{i,j} \in \{0, 1\}$ represents the fraction of traffic of flow $j$ that is forwarded through physical link $i$.

**Prediction Assisted Measurement Algorithm (PAMA):** DeepFlow initially uses the AFPDA algorithm to zoom-in to active flow prefixes that exceed the threshold $\theta$. After AFPDA has detected all the important flow (the process can take more than 1 epoch), DeepFlow employs the PAMA algorithm that starts performing measurements in a round robin fashion. Specifically, PAMA will install measurement rules for the active prefixes in order to collect $d$ samples before start using predictions for the next $T$ epochs for each prefix. So, if the total available memory in the network is $m$, then we can cover up to $\left( \lfloor \frac{T}{d} \rfloor + 1 \right) \times m$ many flows before we start collecting measurements again for the first round of prefixes. This allows DeepFlow to multiplex predictions and measurements and achieve a more fine-grained view of the network.

**Flow-Size Change Detection Algorithm (FSCDA)** In order to make sure that DeepFlow does not produce large estimation errors in case of unexpected traffic anomalies (e.g. network outages, or DDoS attacks), DeepFlow employs periodically an algorithm called FSCDA that aims to detect the set of flows that might have been involved or impacted by a traffic anomaly. For this, during every measurement period, it retrieves port statistics that are available without any explicit measurement rule in each switch, and checks for change-points (e.g. sudden changes in the volume of a time series). If a change is detected in a subset of the port-level time-series, then the flows that pass through all the affected links are calculated since these are the ones that should have been impacted by the network anomaly. So, in the next measurement epoch, DeepFlow will focus on these flows and install exact measurement rules, if they are still operating in prediction mode.

## 4 EVALUATION

In this section, we present the results from the evaluation of DeepFlow and demonstrate that predictions can be effectively used to assist network measurements. For this, we present the results from our evaluation study that use flow size modeling, and other simulations in order to provide fine grained measurements.

The problem of modeling network flow time-series is not new in the relevant literature. Most of the previous works have focused on modeling the aggregate size of a number of flows over time windows of several minutes [22–24]. These models are traditionally good for coarse grain traffic matrix predictions, since they leverage long-range dependencies in order to predict how the overall volume seen by an observation point will behave in the future. On the other hand, there have been some efforts on modeling aggregated flow-sizes in shorter time scales, such as [25–27]. A in-depth discussion of all the previous work on the topic is out of the scope of this paper. Here, we emphasize on the main differences of the existing approaches with DeepFlow and motivate the need of a new effort to accurately model fine grained flows in short time scales. Specifically, most of the prior research was done more than 2 decades ago, with the flow datasets being significantly different compared to now due the significantly lower network speeds, the limited amount of multimedia traffic (e.g. video, VOIP etc.), and the different traffic dynamics overall. Second, in the case of more recent examples such as [28], only aggregated traffic at the link (port) level was modeled in large timescales (i.e. 15 minutes), which differs significantly from what DeepFlow aims to model. For this reason, in DeepFlow we take a different approach, and inspired by the recent advances in Deep Learning, we proceed to test the effectiveness of the state-of-the-art deep learning model for time-series [2, 21] in the field of flow size prediction of recent network logs in short time scales ($\leq$ 5 sec).

A Long-Short-Term-Memory (LSTM) model is a form of a recurrent neural network that has gained popularity in the recent years due to its effectiveness in modeling complex time series with time lags of unknown size that separate important events [21]. The main idea of LSTM is the use of self-loops where the gradient can flow for long durations without vanishing or exploding. This, in combination with the use of a forget-gate, allows the LSTM to accumulate knowledge that can be "forgotten" later depending on the input data. To the best of our knowledge, this is the first time that LSTM models are used for modeling fine-grained network flow sizes in short time scales.

In order to validate the effectiveness of LSTM for network traffic modeling under various traffic conditions, we analyzed real network traces from [20] as well as run simulations in Mininet. Due to the limited space in this publication, we are going to only provide the some representative results that can be used to validate the proposed work.

**CAIDA Traffic Traces**: The dataset in [20] contains anonymized passive traffic traces from CAIDA's high speed passive monitors obtained from a Chicago data-center in 2016. In order to model the CAIDA traffic, we group the traffic per source/destination IP over short time periods of 1 to 5 seconds, and then the source and destination prefixes are aggregated based on the subnet of a given mask size that they belong to, which varies between 1 and 15. This way we are able to model network traffic volume time series at various time and aggregation scales. In addition, AFPDA was shown to converge in 16382 measurements with 1Mbps threshold, while it took 90 seconds to complete, with a measurement of up to /15 flows and an average accuracy of 14%.

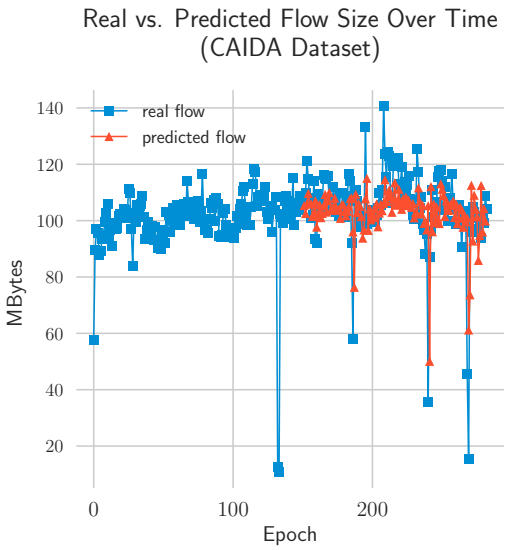


**Figure 3: LSTM predictions of the size of a /15 aggregated flow from CAIDA trace (epoch duration = 5 seconds).**

In Fig. 3, a sample aggregated flow of 300 epochs is shown (epoch size = 5 seconds), as well as its prediction curves. As we can see from the graph, the prediction curve approximates the real flow sizes pretty well, yielding to an average mean absolute percentage error (MAPE) of 12%.

**Mininet Simulations**: In the Mininet scenario, we simulate the network topology of Google B4 [19] with 1 Gbps links, where each OpenVSwitch has 5 hosts attached to it and they are all concurrently sending traffic to a random destination host over their shortest path. In Fig. 4 a sample aggregated flow of 100 epochs is shown (epoch size = 5 second), as well as its prediction curve. As we can see from the graph, the prediction curve approximates the real flow size pretty well, yielding to an average MAPE of 3.9% across all the flows measured, which is much better than the case of CAIDA trace. One reason for this is the fact that in Mininet, the TCP flows created where active for longer periods of time without other interfering traffic such as UDP, thus yielding to more stationary flows that can be modeled with higher accuracy. Finally, AFPDA was shown to converge with 16 measurements at 1Mbps threshold, while that took 5 seconds to complete, while allowing measurement of up to /32 flows with an average accuracy of 4%.

**AFPDA Evaluation**: In order to further evaluate the performance of AFPDA, we proceed to simulate its evolution under various traffic conditions. Specifically, we implemented the 2D splitting for various static and random volume splitting distributions that characterize how the overall volume of a subnet is distributed when the 2D splitting takes place. In the figures presented below, we show the results obtained by using random traffic splitting generated by a Dirichlet distribution, for a network with an aggregate traffic rate of 100Gbps, a flow size threshold of 10Mbps, a maximum subnet mask size of /15. Specifically, Fig.5 shows the percentage of high volume prefixes (i.e. more than the threshold) for each mask size. From the graph we can see that the percentage drops exponentially as we keep splitting further the 2 dimensional IP space, which

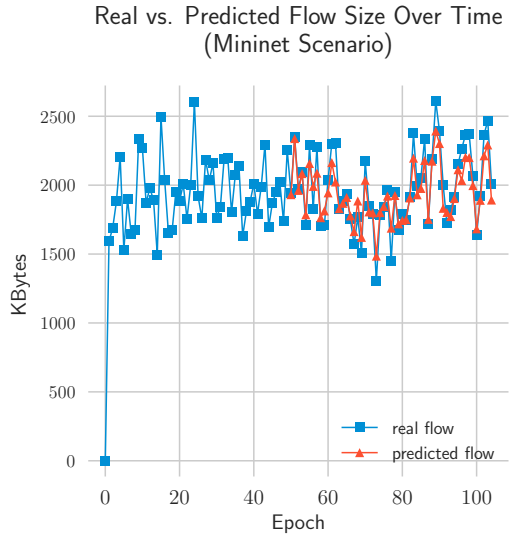


**Figure 4: LSTM predictions of the size of a /32 flow from Mininet simulation (epoch duration = 5 seconds).**

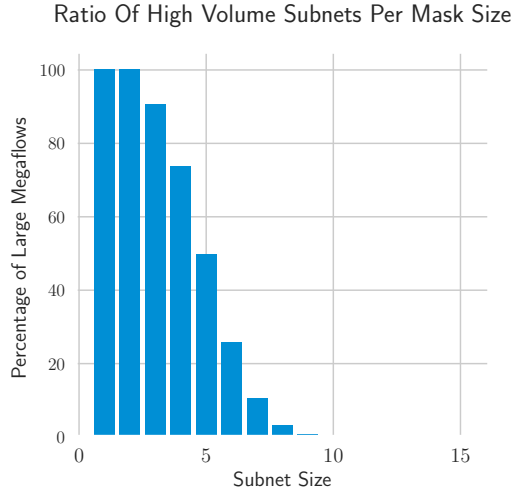


**Figure 5: The percentage of high volume prefixes for each source and destination IP mask size, as determined by AFPDA using random traffic splitting.**

also shows that despite the fact that we have a potentially huge number of prefixes to explore, AFPDA eventually focuses only on few thousands of flows that are important only. This is also evident in Fig.6 where the exact number of prefixes exceeding the threshold is shown for each mask size. From there, we can see that AFPDA converges pretty fast since after mask size 8, the number of large flows decreases exponentially, and AFPDA will skip any sub-prefixes that do not exceed the threshold. Finally, for the experiment shown above, the converge time was 90 seconds, with an epoch size of 5 seconds and an average number of available measurement rules of 500 per epoch.

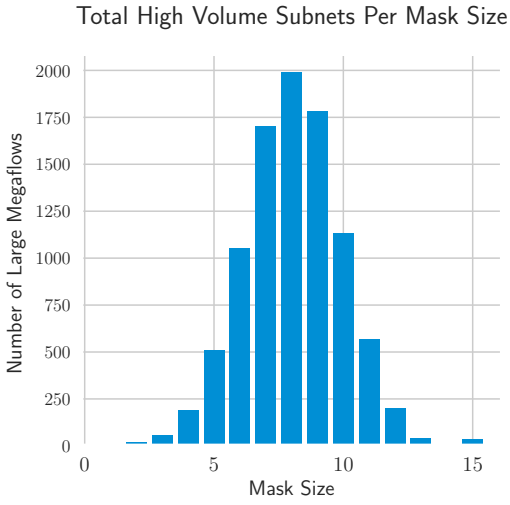


**Figure 6: The total number of high volume prefixes for each source and destination IP mask size, as determined by AFPDA using random traffic splitting.**

## 5 RELATED WORK

Traffic measurements in SDN can be achieved using using two main approaches: a) TCAM-based traffic counters (e.g. [9], [15], [16], [17]), and b) hash-based counters such as sketches (e.g. [12], [13]). In this work, similar to the work in [9] and [16], we focus on the problem traffic measurement in SDN using TCAM-based counters, since they provide immediate deployability in commercial switches. In [9], the authors propose DREAM, a TCAM-based measurement framework that focuses on balancing measurement accuracy and the amount of resources (i.e. TCAM rules) that are used for monitoring specific flows. DREAM is suitable for tasks for which accuracy can be estimated, such as (Hierarchical) Heavy Hitter detection, and change detection, and does not provide a generic framework of TM estimation. In [15], the authors present OpenTM, a framework for TM estimation for OpenFlow networks that is based on simple flow statistics retrieval from SDN switches. The paper assumes that all flow rules fit in the TCAM and thus can be tracked with exact match rules (i.e. no wildcard rules are used). In addition, OpenTM constantly requests flow counters from various switches over the path of a flow, thus generating significant overhead. Finally, the paper compares the performance of various counter retrieval strategies (e.g. random vs last switch vs round robin etc.). In [16], the authors propose OpenMeasure, an extension of iStamp [11] for TM estimation in hybrid SDN deployments that uses an adaptive counter placement mechanism to detect large flows, and then places exact match rules in the TCAM for the large flows detected. The framework uses a simple prediction framework to estimate the size of a flow in the next measurement period and based on that select the target flows to monitor. From the previous work presented above, the most relevant is OpenMeasure [16]. However, there are significant differences compared to DeepFlow as described below: a) OpenMeasure does not provide fine grained flow measurement (it picks only the most important flows that can

fit in the TCAM), b) OpenMeasure does not use predictions to substitute measurements, but instead, it uses a model to estimate the largest flows to monitor with TCAM rules in the next measurement epoch, c) the two models used for estimation are simple linear models that have not been studied for their effectiveness in predicting flow sizes in small time scales (i.e. < 15 minutes) with smaller flow-aggregation ratio, where flows appear to be less steady, d) OpenMeasure is suitable for large measurement epochs with large flow aggregation ratio.

## 6 CONCLUSION AND FUTURE WORK

Providing fine-grained measurement is mandatory for many network applications. In this paper, we propose DeepFlow, a prediction-assisted measurement framework for SDN that uses the available TCAM memory to install measurement rules for important flows, and uses an efficient machine learning algorithm to predict the size of rest of the flows that cannot be monitored with exact match rules, by using historical data from previous measurement periods. For our future work, we are planning to expand DeepFlow to use complex flow interactions and more network signals for further reduce the number of exact flow measurements needed.

## REFERENCES


[1] Albert Mestres, et al. 2017. Knowledge-Defined Networking. SIGCOMM Comput. Commun. Rev. 47, 3 (2017), 2-10.

[2] Sepp Hochreiter and Jurgen Schmidhuber. 1997. "Long Short-Term Memory". Neural Comput. 9, 8 (1997), 1735-1780.

[3] Augustin Soule, Kave Salamatian, and Nina Taft. 2005. "Combining filtering and statistical methods for anomaly detection". In Proc. of IMC '05. USENIX, Berkeley, CA, USA, 31-31.

[4] Matthew Roughan, Mikkel Thorup, and Yin Zhang. 2003. "Traffic engineering with estimated traffic matrices". In Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement (IMC '03). ACM, New York, NY, USA, 248-258.

[5] Theophilus Benson, Ashok Anand, Aditya Akella, Ming Zhang. 2011. "MicroTE: fine grained traffic engineering for data centers". In Proceedings of the 2011 Conference on Emerging Networking Experiments and Technologies (Co-NEXT '11). ACM.

[6] Andrew R. Curtis, et al. 2011. "DevoFlow: scaling flow management for high-performance networks". SIGCOMM Comput. Commun. Rev. 41, 4 (2011), 254-265.

[7] Cristian Estan and George Varghese. 2002. "New directions in traffic measurement and accounting". SIGCOMM Comput. Commun. Rev. 32, 4 (August 2002), 323-336.

[8] A. Yassine, H. Rahimi and S. Shirmohammadi. 2015. "Software defined network traffic measurement: Current trends and challenges," in IEEE Instrumentation & Measurement Magazine, vol. 18, no. 2 (April 2015), 42-50.

[9] Masoud Moshref, Minlan Yu, Ramesh Govindan, and Amin Vahdat. 2014. DREAM: dynamic resource allocation for software-defined measurement. In Proceedings of the 2014 ACM conference on SIGCOMM. ACM, New York, NY, USA, 419-430.

[10] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," in Proceedings of the IEEE, vol. 103, no. 1, pp. 14-76, Jan. 2015.

[11] Malboubi, Mehdi, et al. "Intelligent sdn based traffic (de) aggregation and measurement paradigm (istamp)." Proceedings of IEEE INFOCOM, 2014.

[12] Yu, Minlan, Lavanya Jose, and Rui Miao. "Software-Defined Traffic Measurement with OpenSketch." Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13). 2013.

[13] Moshref, Masoud, et al. "SCREAM: Sketch Resource Allocation for Software-defined Measurement." ACM CoNEXT, 2015.

[14] Liu, Zaoxing, et al. "One sketch to rule them all: Rethinking network flow monitoring with UnivMon." Proceedings of the 2016 conference on ACM SIGCOMM, 2016.

[15] Tootoonchian, Amin, Monia Ghobadi, and Yashar Ganjali. "OpenTM: traffic matrix estimator for OpenFlow networks." International Conference on Passive and Active Network Measurement. Springer Berlin Heidelberg, 2010.

[16] Liu, Chang, AMehdi Malboubi, and Chen-Nee Chuah. "OpenMeasure: Adaptive flow measurement & inference with online learning in SDN." Computer Communications Workshops (INFOCOM WKSHPS), IEEE, 2016.

[17] Gong, Yanlei, et al. "Towards accurate online traffic matrix estimation in software-defined networks." Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research. ACM, 2015.

[18] Van Adrichem, Niels LM, Christian Doerr, and Fernando A. Kuipers. "Opennetmon: Network monitoring in openflow software-defined networks." 2014 IEEE Network Operations and Management Symposium (NOMS). IEEE, 2014.

[19] Jain, Sushant, et al. "B4: Experience with a globally-deployed software defined WAN." ACM SIGCOMM Computer Communication Review 43.4 (2013): 3-14.

[20] CAIDA Anonymized Internet Traces 2016. http://www.caida.org/data/passive/passive_2016_dataset.xml

[21] Ian Goodfellow and Yoshua Bengio and Aaron Courville, "Deep Learning", MIT Press, 2016.

[22] K. Papagiannaki, K. Papagiannaki, N. Taft, N. Taft, Z. Zhang, Z. Zhang, C. Diot, and C. Diot, "Long-Term Forecasting of Internet Backbone Traffic: Observations and Initial Models", vol. 0, no. C, pp. 1178-1188, 2003.

[23] K. U, Z.-L. Zhang, and S. Bhattacharyya, "Profiling internet backbone traffic", ACM SIGCOMM Comput. Commun. Rev., vol. 35, no. 4, p. 169, 2005.

[24] You, C. and Chandra, K., "Time Series Models for Internet Data Traffic", In Proc. of IEEE LCN 1999.

[25] C. Barakat, P. Thiran, G. Iannaccone, C. Diot, and P. Owezarski, "Modeling Internet backbone traffic at the flow level," IEEE Trans. Signal Process, vol. 51, no. 8, pp. 1-12, 2003.

[26] S. Basu and A. Mukherjee, "Time Series Models for Internet Traffic", in 24th Conf. on Local Computer Networks, Oct. 1999, pp. 164-171.

[27] A. Sang and S. Li, "A Predictability Analysis of Network Traffic", in INFOCOM, Tel Aviv, Israel, Mar. 2000.

[28] A. Azzouni and G. Pujolle, "A Long Short-Term Memory Recurrent Neural Network Framework for Network Traffic Matrix Prediction", CoRR abs/1705.05690, June 2017.