An LSTM Framework for Software-Defined Measurement

Aggelos Lazaris, *Member, IEEE,* and Viktor K. Prasanna, *Fellow, IEEE*

***Abstract*—Providing fine-grained traffic measurement is crucial for many network management and optimization tasks such as traffic engineering, anomaly detection, load balancing, power management, and traffic matrix estimation. Software-defined networks can potentially enable fine-grained measurement by providing statistics for each forwarding rule. However, the TCAMs that are used for rule matching and statistics generation have limited size due to their high cost and power consumption. This allows only a fraction of the flows to be monitored. In this paper, we present DeepFlow, a framework for scalable software-defined measurement that relies on an efficient mechanism that a) adaptively detects the most active source and destination IP prefixes, b) collects fine-grained measurements for the most active prefixes and coarse grained for the less active ones, and c) uses historical measurements in order to train a Long Short-Term Memory (LSTM) 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 suffer from low accuracy. An extensive experimental evaluation study using real network traces shows that DeepFlow outperforms the baselines in terms of the total number of flows measured.**

***Index Terms*—Software-Defined Networking (SDN), traffic measurements, machine learning, LSTM, Knowledge-Defined Networking (KDN).**

I. Introduction

THE availability of fine-grained network traffic measurements is necessary in order to provide the information required for a large variety of network management and optimization tasks such as network monitoring, traffic engineering, anomaly detection, network accounting, network analytics, performance diagnostics, load balancing, power savings, and Traffic Matrix (TM) estimation [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11]. Software-Defined Networking (SDN) is an emerging network architecture that decouples the control plane from the forwarding plane. OpenFlow [12] has provided a standard centralized mechanism for a network controller to install forwarding rules at the flow tables and retrieve statistics for a given flow (defined, for example, by a source and destination IP prefix) such as the total bytes or packets transferred. This can enable a wide variety of fine-grained measurement tasks that can increase the visibility into the network, as long as the forwarding rules are not too broad to match multiple flows. However, in practice, the Ternary Content-Addressable Memory (TCAM) memory of the switches that is used for prefix (i.e. wildcard) matching and traffic statistics generation 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 L3/L2 rules (i.e. for IP or MAC addresses based forwarding), which is much smaller compared to the tens or hundreds of thousands of micro-flows that an SDN-enabled switch can concurrently forward [13], [14], [15]. Thus, more efficient mechanisms are needed in order to enable fine-grained traffic measurement.

Prior work on SDN traffic measurement has either assumed specialized hardware support (e.g. sketches) at the switches [16], [17], [18], [58], [57], or it has focused only on specific measurement tasks only such as heavy hitter detection, or anomaly detection [19]. 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 of interest ([14], [20]) 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. [21], [22]), without providing any visibility beyond the size of TCAM. However, such mechanisms cannot be used to provide a universal low-level view of the network.

In this paper, we introduce DeepFlow, a framework for fine-grained software-defined measurement that can be directly deployed to production hardware switches and which leverages scalable machine learning predictions and an efficient active flow detection engine in order to provide visibility into the network traffic. DeepFlow operates at the control layer and leverages four main observations.

1) 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 it traverses throughout the network, and its exact measurement location can be optimized such that the total number of flows measured is maximized (a similar approach can be applied for a 5-tuple flow definition).
2) 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 modeled more accurately by an efficient time series prediction model.
3) If we periodically use flow rate 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 achieving a deeper view of the network.
4) 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 and install more fine-grained measurement rules by correlating the switch port statistics

A. Lazaris and Viktor K. Prasanna are with the Ming Hsieh Department of Electrical Engineering, University Of Southern California, Los Angeles, CA, 90089 USA e-mail: (alazaris@usc.edu, prasanna@usc.edu). This work has been funded by the U.S. National Science Foundation under grant number ACI 1339756 and the Department of Energy (DoE) under award number DE-EE0008003.

data (which are available "for free" in each epoch) with the routing information such that we detect with high probability which flow prefixes caused the traffic change.

Contributions: The main contributions of our proposed work are summarized below:

1) We introduce an algorithm that can adaptively detect active flow prefixes in the network that generate most of the traffic volume and should be monitored separately.
2) We introduce a framework to optimize the measurement location of each flow in order to increase parallelism and speedup the measurement process.
3) We introduce a scalable framework for modeling diverse flow rate time series using Long Short-Term-Memory (LSTM) models (a type of recurrent neural network deep in time) [33] that leverages past measurements or predictions in order to predict future flow rates.
4) In order to make our prediction framework scalable, we propose a feature-based time series clustering framework that groups the flow rate time series into groups with similar characteristics, and then train a single LSTM model per group, instead of a different model for each flow prefix.
5) We introduce a framework for combining predictions with real measurements in order to increase the visibility into the network, while at the same time allow the controller to measure other unexplored areas of the IP space.

One of the main advantages of DeepFlow is that it supports the Knowledge-Defined Networking (KDN) paradigm [23] according to which SDN can be used to enable a knowledge plane on top of the controller which can then be used for traffic predictions and forecasting, as well as other network management tasks that rely on high resolution flow rate data. This makes it quite different than other previous works that either only focus on detecting large flows, or performing a specific measurement task only. In addition, DeepFlow can be used to achieve optimizations results similar to [53] but by applying a data-driven approach at short time scales in order to a) achieve more efficient traffic engineering that can increase the link utilization, b) protect critical traffic from congestion, c) improved load balancing, d) improve admission control, e) reduce power consumption in data center or ISP networks by turning off the equipment (or enabling power saving mode for CPU cores, physical links, VMs, servers, etc.) that is not needed in order to forward or process traffic, and f) improve security even during low volume attacks (due to the improved network visibility). It is important to note here that the high time resolution of our framework is a very important factor that can make a big difference for certain applications like power savings, since the optimal power consumption curve can be better approximated with more granular traffic predictions and thus reduce the power consumed [59].

The rest of the paper is structured as follows: Section II presents the related work. Section III presents some background information and some motivating examples that can better illustrate the applicability of our framework. In Section IV we present the overall architecture of DeepFlow, and in Section V we introduce the concept of prediction-assisted measurement. Section VI presents an analysis of flow time series and how we can use clustering in order to reduce the number of models needed. Section VII presents an experimental evaluation of our proposed work. Finally, Section VIII concludes the paper.

II. Related Work

The problem of network traffic measurement has been extensively studied in the relevant literature in both traditional networks and SDN. Traffic measurement in SDN can be achieved using two main approaches: a) TCAM-based traffic counters (e.g. [19], [14], [21], [22]), and b) hash-based counters such as sketches (e.g. [16], [17], [29], [18], [58], [57], [30], [15]). In this work (a shorter version of which was presented in [26], and [39]), similar to the work in [19] and [21], we focus on the problem of traffic measurement in SDN using TCAM-based counters since it provides immediate deployability in commercial hardware switches. In addition, our previous work in [47] provided a unified framework for overcoming the challenges that switch diversity imposes to network management (including traffic measurement) that can introduce significant delays to the control or data plane and prevent the controller from being able to query the hardware switches for traffic counters. On the other hand, a) hash-based counters are not currently supported in commercial hardware switches, b) they require complex switch hardware upgrades (e.g. ASICs) in order to be implemented, and c) most of them cannot provide a generic fine-grained measurement framework at scale since they consume a lot of CPU cycles, suffer from low accuracy, and cannot always keep up with the line speed. ElasticSketch [57] was designed to achieve elasticity and scale, however it is still not suitable for TCAM based implementations which is our focus here. In addition, our proposed scheme can be adapted to use hash-based measurements as well in its data collection component, and then combine that with its prediction capabilities to increase the overall visibility into the network. Finally, the scope of our work spans mainly the area of Wide Area Networks (WANs), since the large number of flows in such networks pose significant scalability challenges for traffic measurement. However, our framework can also be extended to SDN networks of any scale.

According to [6], measurement frameworks can be classified into one of the following three categories depending on if they provide: a) a balance in overhead implications by using techniques like sampling, aggregation, efficient heuristics etc, b) a resource usage as a trade-off with measurement accuracy, and c) accurate measurements in real-time for decision making. Below, we briefly cover relevant frameworks from all three categories.

The authors in [31] propose iStamp, a scheme for flow measurement that uses a Multi-Armed Bandit algorithm to balance between measuring unimportant flows in aggregate, and important flows with exact monitoring TCAM rules. This way, the authors achieve high monitoring accuracy for the flows of interest. iStamp provides a single switch framework that does not aim to provide a generic measurement solution, neither uses machine learning predictions to enhance the measurement coverage. In [19], 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 where accuracy can be estimated, such as (Hierarchical) Heavy Hitter detection, and change detection,

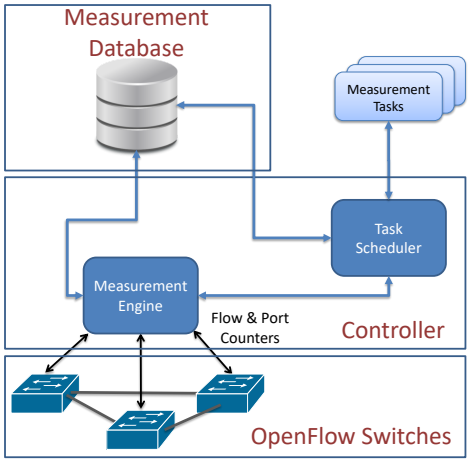


Fig. 1: SDN measurement architecture.

and does not provide a generic framework for fine-grained measurement. In [14], the authors present OpenTM, a framework for TM estimation for OpenFlow networks that is based on simple flow statistics retrieved 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. A very similar approach is also presented in [20] where the frequency of flow counter retrieval is determined depending on the variability of each flow over time. In [21], the authors propose OpenMeasure, an extension of iStamp [31] for measurement in hybrid SDN deployments that uses an adaptive counter placement mechanism to detect large flows in the network, 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 but predictions are not used to substitute measurements. The authors in [22] present a TM estimation framework for hybrid SDN deployments that corresponds to a multi-switch extension of iStamp [31]. The proposed framework does not use online learning (unlike the work in [21]) and monitors flows by de-aggregating aggregated flows in order to provide more fine-grained measurements of important flows. The paper in [54] proposes an adaptive compressive sensing approach for estimating the Origin-Destination TM at the backbone level, using sparse measurements for end-to-end traffic reconstruction. The proposed method is evaluated using the GEANT and ABILENE datasets that provide TM at coarse-grained time scales. The paper in [34] uses an LSTM model to model aggregated traffic at the link (port) level in large timescales (i.e. 15 minutes), which differs significantly from our proposed work that focuses on short time scales (i.e. $\leq$ 30 seconds) and models data at the network prefix level. Finally, the papers in [28], and [41] present neural network approaches for TM estimation in traditional IP networks (i.e. not SDN) where simple link measurements (i.e. SNMP) are used to estimate the Ingress-Egress router TM (i.e. aggregated flow-size estimation).

From the previous works presented above, the most relevant is OpenMeasure [21]. However, there are significant differences compared to our proposed work 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 and focuses only on that), b) OpenMeasure does not use predictions to substitute measurements, but instead, it uses predictions to find the largest flows to monitor with TCAM rules in the next measurement epoch, c) the two models used for prediction are simple linear models that have not been studied for their effectiveness in predicting flow-rates in small time ranges (i.e. $<$ 15 minutes) with smaller flow-aggregation ratio, where flows appear to be more volatile, and d) OpenMeasure is suitable for large measurement epochs (unlike DeepFlow which operates at $\leq$ 30 seconds timescale) with large flow aggregation ratio.

Finally, our previous work in [26] presents an introductory exploration of the main ideas behind DeepFlow, namely the overall architecture and its proof of concept implementation using two small network datasets. In addition, our work in [39] introduces the concept of network time series clustering for model selection which is applied to various network prefixes to assess its modeling effectiveness. On the contrary, in this paper, we present an in-depth analysis of the main concepts behind DeepFlow as well as the idea of using LSTM predictions to substitute measurements.

TABLE I: Notation

Symbol	Description
N	Total number of switches
K	Total number of flows
L	Total number of physical links
f_i	Flow i
s_j	Switch j
M_j	The total memory at switch j
m_j	The total free memory at switch j
w	The number of inputs required to generate a prediction
T	The prediction horizon of the flow model
$\mathcal{M} = (\theta, d_s, d_t)$	A measurement task with minimum flow rate threshold θ, maximum source destination IP prefix length d_s, and maximum destination IP prefix length d_t

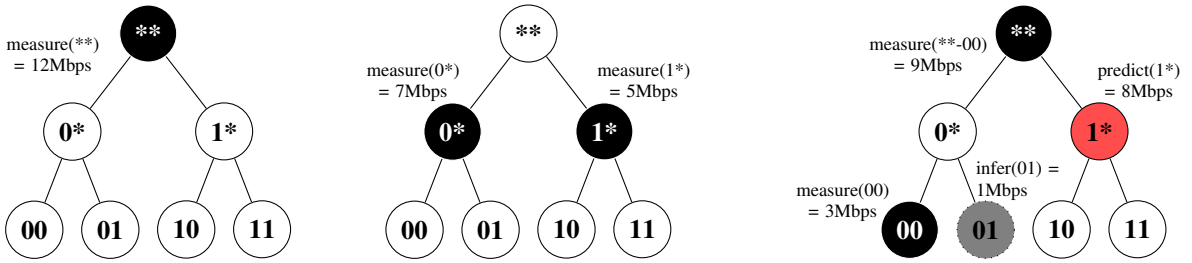
III. Background & Motivation

According to the definition in [52] *"Network management includes the deployment, integration, and coordination of the hardware, software, and human elements to monitor, test, poll, configure, analyze, evaluate, and control the network and element resources to meet the real-time, operational performance, and Quality of Service requirements at a reasonable cost"*. In this work, we focus on the measurement aspect of network management that can enable the in-depth monitoring of the network while satisfying the requirements for real-time operation and reasonable cost.

In the section below, we provide some background information that is necessary in order to introduce our proposed measurement framework. In addition, Table I provides an overview of all the major notation used throughout this paper.

A. Measurement In Software-Defined Networks

SDN switches that support the OpenFlow specification [12] can provide a variety of traffic counters for each flow table,



(a) 1 TCAM rule used (not fine-grained measurement granularity)

(b) 2 TCAM rules used (not fine-grained measurement granularity)

(c) 2 TCAM rules (the leaf node with higher priority) and 1 prediction used (more fine-grained measurement granularity)

Fig. 2: An example of source IP prefix trie with three potential measurement architectures that provide various granularities. When predictions are used, the granularity level can be increased for the same TCAM utilization.

flow entry, port, queue, group, group bucket, meter and meter band. The counters primarily measure the total number of bytes and packets received or transmitted, as well as the duration (in seconds) that these counters correspond to. The counters can be retrieved by the SDN controller depending on the needs of each measurement task. From the list of counters provided above, in this work we focus only on the flow entry and port counters only, since a) these are the most fundamental ones that are supported by all the major switch vendors, and b) they do not require any complex switch configuration in order to be implemented. The overall SDN measurement architecture is shown in Fig. 1, where a measurement task is requested through the controller application, and the task scheduler schedules the rule installation and counter retrieval through the measurement engine. The retrieved traffic counters are then stored in a measurement database.

Even though the basic idea behind SDN measurement is simple, there are various measurement methodologies one can use with a wide range of complexities. To get a better idea, we illustrate in Fig. 2 three examples of flow measurements with a) low TCAM utilization, b) higher TCAM utilization, and c) higher TCAM utilization and use of flow predictions. Specifically, in Fig. 2 (a) we illustrate a 2-bit source prefix trie, and assume that the forwarding rules at the SDN switch are of the form {`src_ip=(**), dst_port=1`}. In such a case, the statistics the switch stores about the single forwarding rule can cover up to 4 micro-flows (i.e. 00-11) and are not fine-grained. In Fig. 2 (b), the same rule space from (a) is represented by partitioning the root prefix in the following two: {`src_ip=(0*), dst_port=1`} and {`src_ip=(1*), dst_port=1`}. In such a case, with two mega-flow rules we can achieve higher granularity since now each rule will store statistics matching two micro-flows (i.e. 00, 01, and 10, 11). In Fig. 2 (c), the same rule space from (a) is represented by combining the high priority micro-flow rule {`src_ip=(00), dst_port=1, priority=0`}, with a low priority root prefix rule {`src_ip=(**), dst_port=1, priority=1`}. This way, only the traffic matching rule 00 will be counted in the measurement of the first rule, whereas the rule at the root prefix will count everything but 00. As we can see from this, if we are able to enhance the measurement process with predictions for prefix 1* (as an example) that models two micro-flows, then the size of flow 01 can be inferred using simple flow algebra, thus leaving only two out of the 4 micro-flows without exact low level measurement. The process described above can be generalized as we can see in the following sections in the 2 dimensional prefix plane (i.e. source and destination IP), and depending on the prediction capabilities we can get, we can achieve any desired granularity level. In addition, a spatial component can be added to the process to capture the location that a flow can be measured at (i.e. switch), which can allow more flows to be measured concurrently for a given number of available TCAM rules.

B. Flow Model

Definition 1. A flow is defined as a set of IP packets passing an observation point in the network during a certain time interval which share a set of common properties such as the source and the destination IP address or prefix.

Definition 2. A flow-rate time series $F_i = f_i^{(1)}, f_i^{(2)}, ..., f_i^{(n)}$ is an ordered set of n real-valued variables that correspond to the total traffic volume of flow f_i over a measurement interval.

Definition 3. Given a time series F_i of length n, a subsequence $F_i^{(p)}$ of F_i is a sampling of length $w < n$ of contiguous positions from F_i, that is, $F_i^{(p)} = f_i^{(p-w+1)}, f_i^{(p-w+2)}, ..., f_i^{(p)}$ for $w \leq p \leq n$.

C. Traffic Matrix Estimation

A Traffic Matrix (TM) represents the volume of traffic flowing through all possible Origin-Destination (OD) pairs in a network. TM can be inferred using partial link measurements as well as other statistical methods that can replace missing information such as compressive sensing [7], [54]. In traditional networking, SNMP measurements are frequently used to obtain link statistics that can be combined with routing information to infer the actual flow sizes to some extend. More formally, let $G(\mathcal{S}, \mathcal{L})$ be the graph representing the network topology where $\mathcal{S}$ is the set of switches and $\mathcal{L}$ is the set of physical links in the network. Also, let $N = |\mathcal{S}|$ and $L = |\mathcal{L}|$ the total number of switches and links, respectively, and K the total number of flows in the network where the volume of flow f_i at time period t is denoted as $f_i^{(t)}$ or for simplicity x_i. The routing in G is assumed to be known and represented by the $L \times K$ matrix $\mathbf{R}$ each element $r_{i,j}$ of which corresponds to the fraction of traffic of flow j going through link i. If we represent all the flow sizes x_i's as a vector $\mathbf{x}$ and the

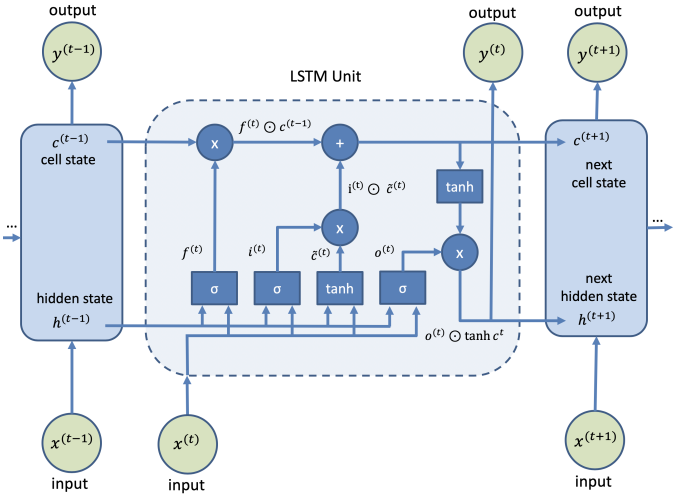


Fig. 3: Architecture of an LSTM unit that connects to other units of a recurrent neural network.

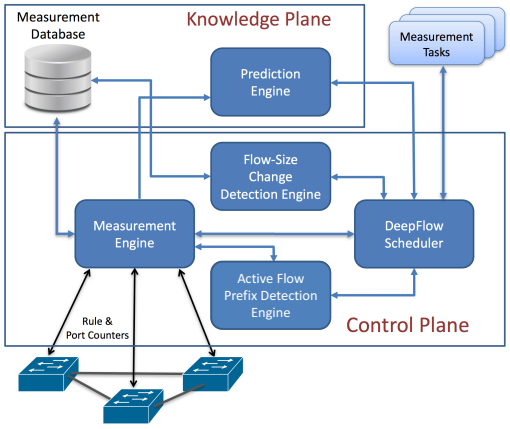


Fig. 4: DeepFlow architecture components.

set of link measurements y_j's as a vector $\mathbf{y}$, then the traffic matrix estimation problem is described by the system of linear equations $\mathbf{R} \cdot \mathbf{x} = \mathbf{y}$ that has L equations and K unknowns. Since the above linear system is ill-posed (i.e. there are far less links than micro-flows in the network), our proposed work can be used to add more equations to the linear system above by a) using the additional flow counters from OpenFlow SDN switches and their ports, and b) using machine learning predictions that can estimate some of the x_i's, or partial sums of them (e.g. instead of x_1 and x_2 we might be able to know $x_1 + x_2$ etc.).

D. Long Short-Term Memory Models

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 [27]. The main idea behind 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. LSTMs are characterized by the following recursive equations:

$$\mathbf{f}^{(t)} = \sigma\left(\mathbf{W}_f \mathbf{x}^{(t)} + \mathbf{U}_f \mathbf{h}^{(t-1)} + \mathbf{b}_f\right) \quad (1)$$

$$\mathbf{i}^{(t)} = \sigma\left(\mathbf{W}_i \mathbf{x}^{(t)} + \mathbf{U}_i \mathbf{h}^{(t-1)} + \mathbf{b}_i\right) \quad (2)$$

$$\tilde{\mathbf{c}}^{(t)} = \tanh\left(\mathbf{W}_c \mathbf{x}^{(t)} + \mathbf{U}_c \mathbf{h}^{(t-1)} + \mathbf{b}_c\right) \quad (3)$$

$$\mathbf{c}^{(t)} = \mathbf{i}^{(t)} \odot \tilde{\mathbf{c}}_{(t)} + \mathbf{f}^{(t)} \odot \mathbf{c}^{(t-1)} \quad (4)$$

$$\mathbf{o}^{(t)} = \sigma\left(\mathbf{W}_o \mathbf{x}^{(t)} + \mathbf{U}_o \mathbf{h}^{(t-1)} + \mathbf{b}_o\right) \quad (5)$$

$$\mathbf{h}^{(t)} = \mathbf{o}^{(t)} \odot \tanh(\mathbf{c}^{(t)}) \quad (6)$$

where $\mathbf{f}^{(t)}, \mathbf{i}^{(t)}, \tilde{\mathbf{c}}^{(t)}, \mathbf{c}^{(t)}, \mathbf{o}^{(t)}, \mathbf{h}^{(t)}, \mathbf{x}^{(t)}, \mathbf{b}_f$ are the forget gate, input gate, candidate state, current state, output gate, hidden state, and input data, respectively, $\mathbf{W}_f, \mathbf{W}_i, \mathbf{W}_c, \mathbf{W}_o$ are the input weights for the forget gate, input gate, candidate state gate, and output gate, respectively, and $\mathbf{U}_f, \mathbf{U}_i, \mathbf{U}_c, \mathbf{U}_o$ are the recurrent weights for the forget gate, input gate, current state, and output gate, respectively. In addition, $\odot$ is the (element-wise) Hadamard product, σ is the sigmoid function, and $\mathbf{b}_f, \mathbf{b}_i, \mathbf{b}_c, \mathbf{b}_o$ are the bias terms for the forget gate, input gate, candidate state, and output gate, respectively. The architecture of an LSTM recurrent neural network is shown in Fig. 3 from where we can see the internal structure of the LSTM unit at time t, as well as how it connects to past and future states to form a recurrent neural network.

IV. DeepFlow Design Overview

Based on the ideas presented in the previous section, we developed DeepFlow, a software-defined measurement framework that uses predictions to enhance the measurement process whenever exact measurements are not feasible, and can be used for any application that leverages fine-grained traffic measurement. In this section, we present the main components of DeepFlow, as shown in Fig. 4. In the analysis presented below, a flow is defined as the traffic between a source and destination IP prefix. However, the same concepts can be applied to higher dimensions (e.g. 5-tuples).

Measurement Task: In order to start collecting DeepFlow measurements, the network operator needs to specify a measurement task with 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 θ 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 θ can be also defined in terms of the total available bandwidth in the switches, e.g. 0.001% of the total bandwidth. More formally, we can define a measurement task as follows:

Definition 4. A *measurement task* $\mathcal{M}$ is defined as the process of collecting flow-rate information about all the flows in the network that have a rate of at least θ, and a source/destination IP prefix size of at most d_s/d_t, respectively. So, a task can be characterized by the 3-tuple $\mathcal{M} = (\theta, d_s, d_t)$

Prediction Engine & Measurement Database: DeepFlow's prediction engine uses historical data stored in the cloud in a measurement database (a data 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 administrator, 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.

V. Prediction-Assisted Measurement

A. Active Flow Prefix Detection

One the of main challenges in SDN measurement 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, we developed the Active Flow Prefix Detection Algorithm (AFPDA) that operates on the 2-Dimensional IP space (i.e. source and destination IP prefix) and iteratively detects what are the most active prefixes in the network. 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 AFPDA is to start the flow zoom-in process where it tries to locate source/destination IP prefixes that have large

Algorithm 1: ActiveFlowPrefixDetection

```
Data: d_s, d_t, θ, F_w // task & forwarding rules
Result: List of active leaf nodes L_a
F_measure ⟵ ∅ // flows to measure
F_install ⟵ ∅ // flow counters to install
T = PrefixTrie(F_w)// convert rules to trie
while True do
    for p ∈ T.GetCurrentRoots() do
        s = p.state
        if s == "measurement" then
            F_measure ⟵ F_measure ∪ {p}
        if p.src.mask_length ≤ d_s &&
          p.dst.mask_length ≤ d_t then
            if s == "unmeasured" then
                F_install ⟵ F_install ∪ {p}
                p.state = "to_be_measured"
    F_location ⟵ CounterILP(F_install)
    for switch, prefix in F_location do
        InstallRule(switch, prefix)
        T[prefix].state = "measurement"
        T[prefix].switch = switch
    for prefix in F_measure do
        traffic = CollectMeasurements(prefix.switch,
          prefix.rule)
        prefix.traffic.append(traffic)
        if traffic < θ then
            prefix.state = "terminal"
        else
            prefix.isActive = True
            prefix.isRoot = False
            sub_prefixes = SplitPrefix(prefix, d_s, d_t)
            for sub_prefix in sub_prefixes do
                sub_prefix.state = "unmeasured"
                // to trigger new measurements
                   starting from this node
                sub_prefix.isRoot = True
return GetActiveLeafNodes(T)
```

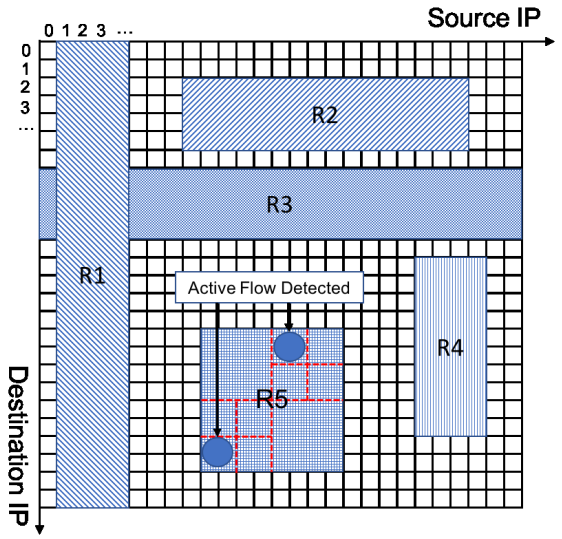


Fig. 5: An example of the set of rules at a switch, represented as shaded areas in a 2-dimensional plane. The red (dashed) lines in R5 correspond to the rule splitting AFPDA performs in order to detect the two active flows in the area of R5 (i.e. the two circles).

volumes and need to be further split into longer active prefixes that can be measured separately. For this, AFPDA 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. So, part of the rule's matching traffic will be offloaded to the new sub-rule, thus allowing more fine-grained measurement. By following the steps outlined in Algorithm 1, AFPDA splits the source/destination prefix plane into smaller regions iteratively until further splitting measures flows with smaller volume than the minimum threshold θ, where it stops or when the maximum prefix lengths d_s, d_t have been reached (defined by the input task). If no threshold θ has been specified, the process will stop when the maximum prefix lengths for the source and destination prefixes have been reached. This process is illustrated in the example of Fig. 5 where the rule space for rule R5 is split twice until two flows (the two circles) were detected which exceeded the threshold θ with a given maximum zoom-level of `/31`.

B. Optimizing The Measurement Location

In order to find the optimal location that a monitoring rule should be installed while maximizing the total number of flows monitored, we proceed to formulate the problem as an Integer Linear Program (ILP) that needs to be solved before installing monitoring rules at the switches.

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 (potentially 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:

$$\text{minimize} \quad \sum_{j=1}^{N} \left(m_j - \sum_{i=1}^{K} x_{i,j} \right) \tag{7}$$

$$\text{subject to} \quad \sum_{i=1}^{K} x_{i,j} \leq m_j, \ j = 1, \ldots, N \tag{8}$$

$$\sum_{i=1}^{N} x_{i,j} \leq 1, \ j = 1, \ldots, K \tag{9}$$

$$x_{i,j} \in \{0, 1\}, \ i = 1, \ldots, N; \ j = 1, \ldots, K \tag{10}$$

$$x_{i,j} \leq r_{i,j}, \ i = 1, \ldots, N; \ j = 1, \ldots, K \tag{11}$$

In the above linear program, Eq. 7 maximizes the total number of rules monitored. Eq. 8 guarantees that we will not try to install more rules to a switch than its available memory. Eq. 9 forces the number of rules per flow to be equal to one (i.e. no flow will be monitored in two switches). Eq. 10 makes sure that the auxiliary variable $x_{i,j}$ is binary, and Eq. 11 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 routing of flow j through switch i.

In the above analysis, we assume that if an aggregated flow gets split at some switch throughout its path, or two or more flows get merged at some switch throughout their path, then we can follow one of the options below, before solving the ILP in Eq. 7 - 11: a) split a flow into its most fine-grained micro-flows that exist as forwarding rules at the switches and use rules that match the micro-flows in the ILP, or b) monitor the flows only at their aggregation switch without providing further granularity. However, the best option would really depend on the application scenario, and the total number of micro-flows that appear as separate forwarding rules in the TCAMs. A similar approach applies in the case of multiple ingress switches per IP prefix, in which case we can flag the flows with a switch-specific tag (e.g. using a VLAN tag or MPLS label) that is used to differentiate between portions of the traffic of a given prefix that is measured on a given location.

C. Using Predictions To Enhance Measurement

AFPDA can be periodically used to find active flow prefixes that exceed the threshold θ. After AFPDA has detected all the important flows (the process can take more than 1 epoch), the multiplexing of measurements and predictions can take place. For this, we have implemented the Prediction Assisted Measurement Algorithm (PAMA) that performs measurements in a round robin fashion (using the same ILP described above) to collect ground truth data that can be used to predict future values. Specifically, PAMA will install measurement rules for the active prefixes in order to collect w samples before starting to generate 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 $\left(\lfloor \frac{T}{w} \rfloor + 1\right) \times m$ many flows before we start collecting measurements again for the first round of prefixes. This allows the system to multiplex predictions and measurements and achieve a more fine-grained view of the network. Below, we provide an analysis of the PAMA for a single switch as well as a multi-switch scenario.

Single Switch Model: Let w be the number of past observations used to predict future flow rates, and let T be the number of future predictions provided by the model (using a rolling time window approach). Let m be the total available TCAM at the switch of interest. Then, in time w we will be measuring the same m flows to collect their w measurements that will be used to predict their sizes for the next T epochs. In the meanwhile, in the interval $[w, w+T]$, we will have also collected w measurements for $\lfloor \frac{T}{w} \rfloor \times m$ more flows. So, overall in time $w + T$ we can measure and generate predictions for $\left(\lfloor \frac{T}{w} \rfloor + 1\right) \times m$ flows. This means that, in each measurement epoch, we will be collecting measurements for m flows, and generating predictions for $\lfloor \frac{T}{w} \rfloor \times m$. So, if we have a set of K important flows to cover (i.e. measure or generate predictions for), it will be:

$$\left(\lfloor \frac{T}{w} \rfloor + 1\right) \times m = K, \tag{12}$$

which shows that the prediction horizon T can be set to allow any coverage of interest. An alternative approach could also be to adjust the significance threshold θ such that the number of significant flows K is reduced.

Multi-Switch Model: In case of multiple switches, the overall benefit from multiplexing predictions can be increased by allowing flows to be measured in any of the switches they traverse. However, the exact number of flows that can be

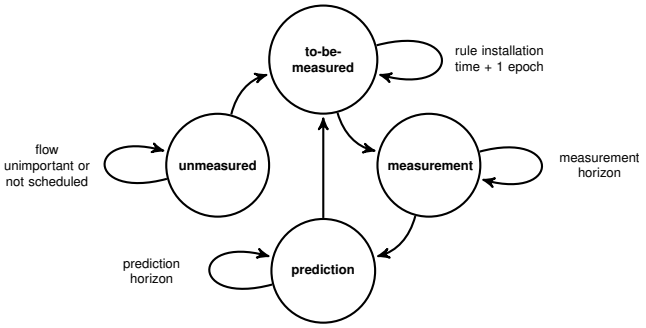


Fig. 6: A finite state machine representation of the states of a given flow prefix that is used by the Prediction Assisted Measurement Algorithm (PAMA).

covered in a given epoch really depends on the routing matrix in the network, since not all the flows can be measured in any of the N switches in the network. So, assuming switch i has m_i free rules, and by using the counter placement ILP to get the optimal measurement location of each flow, we can collect w measurements from up to m_i flows on a given switch and generate predictions for T epochs for each one of them. This allows up to:

$$\sum_{i=1}^{N} \left(\lfloor \frac{T}{w} \rfloor + 1 \right) \times m_i \leq \left(\lfloor \frac{T}{w} \rfloor + 1 \right) \times m \tag{13}$$

flows to be covered, where m is the total number of rules available across all the switches.

From the analysis above it is evident that given a (fixed) routing matrix and a set of significant flows $\mathcal{F}$, in order to cover all the significant flows detected, the prediction horizon needs to be potentially extended for some flows. This also leads to the definition of the "completed task" as follows:

Definition 5. A measurement task that started to be executed at time t_1 is considered *completed* at time $t_2 \geq t_1$ if t_2 is the time when a) all the active flow prefixes have been detected using AFPDA, and b) for every active flow, we have collected at least w samples. Its *completion delay* will then be $t = t_2 - t_1$.

In order to provide a generic framework that can work in a multi-switch scenario and be able to handle any set of input parameters, we proceed to implement PAMA as a Finite State Machine (FSM) that maintains a state for each prefix with the following values: "unmeasured", "to_be_measured", "measurement", and "prediction", as illustrated in Fig. 6. The "unmeasured" state characterizes prefixes that have not been visited yet by the algorithm. The "to_be_measured" state characterizes prefixes that we need to install a measurement rule for, and which will produce measurement data in the next epoch. The "measurement" state characterizes prefixes that have available measurement data to collect, and the "prediction" state characterizes prefixes that we will have to use predictions for during the current measurement epoch. The algorithm also uses a time-to-live (i.e. ttl) metric for each node of the FSM to keep track of the regression window w and prediction window T. The pseudocode of PAMA is shown in Algorithm 2 below.

Algorithm 2: PredictionAssistedMeasurement

```
Data: F_w, θ, d_s, d_t, w, T, period
Result: Flow prefix measurements and predictions
current_epoch = 0
while True do
    F_measure ⟵ ∅ // flows to measure
    F_install ⟵ ∅ // flow counters to install
    F_predict ⟵ ∅ // flows to predict
    if current_epoch % period == 0 then
        active_flows =
          ActiveFlowPrefixDetection(F_w, d_s, d_t, θ)
        flow_prefix_stats = InitializeFlowStats(active_flows)
    for flow_id in active_flows do
        f = flow_prefix_stats[flow_id]
        state = f.cur_state
        if state == "unmeasured" then
            F_install ⟵ F_install ∪ {f}
            f.state = "to_be_measured"
        if state == "to_be_measured" then
            F_install ⟵ F_install ∪ {f}
        if state == "measurement" then
            F_measure ⟵ F_measure ∪ {f}
            if f.ttl == 0 then
                f.state = 'prediction'
                f.ttl = T
        if state == 'prediction' then
            F_predict ⟵ F_predict ∪ {p}
            if f.ttl == 0 then
                f.state = "to_be_measured"
            f.ttl = f.ttl - 1
    F_location ⟵ CounterILP(F_install)
    for flow_id, switch, prefix in F_location do
        InstallMeasurementRule(switch, prefix)
        flow_prefix_stats[flow_id].state = "measurement"
        flow_prefix_stats[flow_id].ttl = w
        flow_prefix_stats[flow_id].switch = switch
    for f in flows_to_collect_measurements do
        f.traffic.append(CollectMeasurements(switch, f.rule))
    for flow in flows_for_predictions do
        f.traffic.append(GeneratePrediction(flow))
    current_epoch += 1
```

VI. Time Series Clustering For Model Selection

The problem of modeling network flow time series or other behavioral aspects of the network 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 [35], [36], [37], or model other metrics such as the rate of in/out calls in a cellular base station [55], [56]. These models are traditionally good for coarse grained 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 to model aggregated flow-sizes in shorter time scales, such as [44], [45], [46]. An in-depth discussion of all the previous work on the topic is out of the scope of this paper. However, 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 two decades ago, with the flow datasets being significantly different compared to now due to 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 [34], [51], only aggregated traffic at the link (i.e. port) level was modeled in large timescales (i.e. 15 minutes) using old datasets, and the analysis at smaller time-scales only used artificial data from two virtual machines, 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 [33], [27] when applied to flow rate prediction using recent network traffic logs at short time scales (i.e. $\leq$ 30 seconds). However, modeling network traffic using a single model has been traditionally a very challenging task, and using LSTM does not make things any easier [39]. On the other hand, using a different model for each flow prefix would not scale due to the large number of possible source - destination IP prefix pairs. So, in order to be able to use a small set of models for modeling a large variety of flow prefixes with different dynamics, we propose the use of a time series clustering framework that assigns time series into a set of disjoint clusters and trains a single machine learning model for each cluster. The idea of clustering network time series for modeling purposes was first introduced in [39] and is briefly presented here to better illustrate the design of DeepFlow.

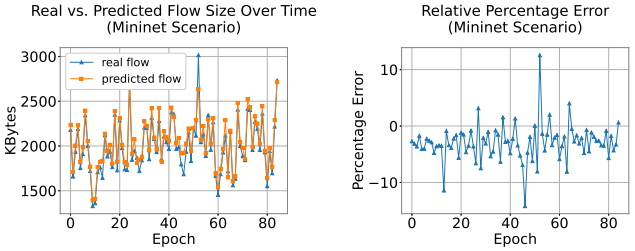
The goal of the time series clustering framework is to find a partition $A_1, A_2, \ldots A_k$ of the set of all the historically active flows F_i for $i \in \{i, \ldots, K\}$ where k is the total number of clusters and K the total number of flows. Let $z_i \in \{1, 2, \ldots, k\}$ be the cluster to which time series F_i is assigned. Then, the clustering algorithm needs to find the optimal $z_i^* = argmax_z\ p(z_i = z|\mathbf{x}_i, \mathcal{D})$ where $\mathbf{x}_i$ is the feature vector that characterizes the time series F_i, and $\mathcal{D}$ is the rest of the input data. In the analysis above, the number of clusters k can be set manually (e.g. in k-Means) or derived automatically by the clustering algorithm (e.g. in DBSCAN).

Time series clustering can be conducted using three kinds of approaches [50]: a) raw data based methods, b) feature-based methods, and c) model-based methods. Here, we propose the use of a feature-based method to cluster the time series F_i since a) this type of approaches work well for heterogeneous time series, b) they require a smaller set of dimensions compared to raw-data approaches, and c) they do not depend on a specific model of the data. However, the challenge here is to find a set of uncorrelated features that can work well overall. For this, we use various features that characterize the dynamics of a flow such as the mean, variance, autocorrelation for various lags, the subnet size, the burstiness of the traffic, the entropy of the distribution of all the flow values, and remove any correlated features found. The full list of features used is shown in Table II. The rationale for choosing these features is that a) they capture well the diverse dynamics of the various network time series, b) they produce relatively good clusters, as indicated by various cluster quality metrics such as homogeneity, completeness, and silhouette coefficient.

The clustering algorithms used are k-Means and DBSCAN.

TABLE II: Features for network traffic time series clustering.

Feature Name	Description
mean	mean traffic size
variance	variance of traffic size
median	median of traffic size
min	the smallest traffic size seen
max	the largest traffic size seen
ptp	the range of values (peak-to-peak)
skew	skewness of the time series
kurtosis	fourth central moment divided by the square of the variance
acf1	lag-1 autocorrelation coefficient
acf10	lag-10 autocorrelation coefficient
entropy	sample entropy of the data
mask	the mask size used to aggregate the traffic
epoch	the epoch size used to aggregate the traffic over time



(a) Actual flow rates vs LSTM predictions.

(b) Relative Percentage Error

Fig. 7: LSTM predictions and relative percentage errors for a /32 flow from Mininet simulation (epoch = 5 sec).

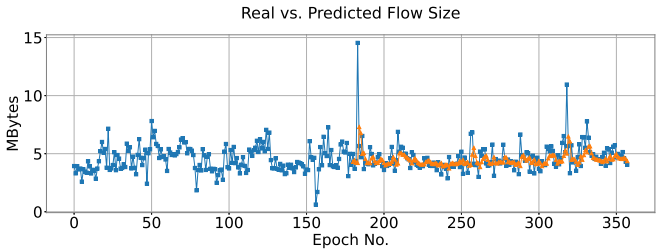
For k-Means, we specify the number of clusters k that needs to be used to partition the time series features such that the within-cluster sum of squares is minimized as follows: $z_i^* = argmin_z \|\mathbf{x}_i - \mu_z\|_2^2$ where μ_z is the cluster's center and it is defined as $\mu_z = \frac{1}{N_k} \sum_{i:z_i=z} \mathbf{x}_i$ and N_z the total number of points in cluster z. Various cluster sizes ranging from 2 to 32 were tested with 20 providing the best results, but depending on the data we want to model this can be tweaked accordingly. On the other hand, DBSCAN only needs to be specified the maximum distance between two samples for them to be considered as belonging to the same neighborhood, and also the minimum number of samples (10 was used as a minimum) in a neighborhood for a point to be considered as a core point, including the point itself.

VII. Evaluation

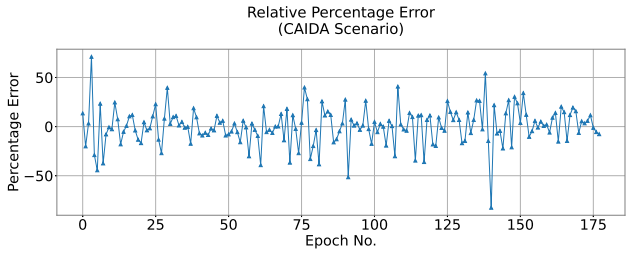
In this section, we present the results from the evaluation of DeepFlow and demonstrate that predictions can be effectively used to assist network traffic measurements.

A. Network Traffic Modeling Using LSTMs

In order to evaluate the hypothesis that flow prefixes can be modeled using LSTMs, we a) analyze data from a real backbone network collected by CAIDA in 2016 [25], and b)



(a) Actual flow rates vs LSTM predictions.



(b) Relative Percentage Error

Fig. 8: Predictions of the size of a /7 aggregated flow from the CAIDA trace (epoch duration = 10 seconds) using VLSTM and its relative percentage errror. The predictions line (orange) approximates the real (blue) very well.

implement a custom network topology in Mininet [38] and collect network logs at the micro-flow level. The network traces from CAIDA were "replayed" in Mininet by following a methodology similar to the one in [19], where subsets of all the IP prefixes in the trace are assigned to specific switches over the B4 topology [24] that forwards the traffic between each source and destination pair. To evaluate the model effectiveness, we calculate the Mean Absolute Percentage Error (MAPE) across all the time series, as defined below:

$$MAPE = \frac{100}{n}\sum_{t=1}^{n}\frac{|f_i^{(t)} - u_i^{(t)}|}{|f_i^{(t)}|} \quad (14)$$

where $f_i^{(t)}$ is the actual flow-rate value and $u_i^{(t)}$ the estimated flow-rate value for a given flow i.

We implement the following variations of LSTM:

1) **Vanilla LSTM (VLSTM):** This is a simple LSTM architecture with an LSTM layer with 50 units, followed by a dense layer with 50 units, dropout of 20%, look back window 3, 20 training epochs (not to be confused with the aggregation epoch used during the dataset creation), batch size 8, and standard scaler on the time series data.
2) **Cluster LSTM (CLSTM):** This is an LSTM architecture that consists of 20 individual LSTM models (equal to the number of clusters that we found to be working well) that are trained using data from a given cluster that the time series are grouped into. Each model has an LSTM layer with 50 units, followed by a dense layer with 50 units, dropout of 30%, look back window 3, 20 training epochs, batch size 128, and standard scaler on the time series data.
3) **Cluster Delta LSTM (CDLSTM):** This is exactly the same architecture as in 3) above, with the only difference that the input data have been pre-processed to calculate their deltas.

The hyper-parameters for each of the models above were chosen using random search, a process that tries out random combinations of parameters from a discrete set, and keeps the one performing the best.

For the case of CAIDA dataset, we implemented both K-Means and DBSCAN in order to cluster the diverse flows into groups with similar characteristics (this was not needed in the case of Mininet scenario since the traffic was artificial with low variability across time series). The optimal number of clusters found was 20, with K-Means producing better clusters overall compared to DBSCAN.

Mininet Simulations: In the Mininet scenario, we simulate the network topology of Google B4 [24] 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. 7(a) a sample aggregated flow of 80 epochs is shown (epoch size = 5 second), as well as its prediction curve. In Fig. 7(b), we show the relative percentage difference between the two time series. As we can see from the graps, the prediction curve approximates the real flow size pretty well, yielding to an average MAPE of 3.9% across all the flows measured. 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 very high accuracy. Another reason is the fact that these models were trained using individual time series which they could specialize on the individual characteristics of each time series. However, since modeling time series individually is not a scalable approach (despite its interesting potentials), we proceed to analyze the CAIDA dataset using our proposed clustering framework since it contains a lot of diverse time series.

CAIDA Dataset: The dataset in [25] contains 1-hour long anonymized passive traces from the CAIDA "equinix-chicago" monitor which is located at the Equinix datacenter in Chicago, IL, and is connected to a backbone 10GigE link of a Tier1 ISP that connects Chicago, IL and Seattle, WA. In order to model the CAIDA traffic, we group the traffic per source/destination IP over short time periods of 5 to 30 seconds, and then the source and destination prefixes are aggregated using various mask sizes that vary from 1 to 7. In Fig. 8(a), we show a flow time series aggregated at the /7 level and 10 sec measurement epoch and its LSTM prediction. In Fig. 8(b) we show the relative error of the predictions over time. The LSTM model used was trained using past measurements from the specific prefix, and as we can see it achieves very good modeling results with a MAPE of 16%.

In order to assess the scalability and efficiency of our proposed clustering framework, we run K-means clustering using all the time series derived from the CAIDA dataset across different measurement epochs and mask sizes, and then train 20 LSTM models using data from one cluster at a time. As we can see from Fig. 9, clustering provides pretty good accuracies with average MAPEs below 20% overall. Vanilla

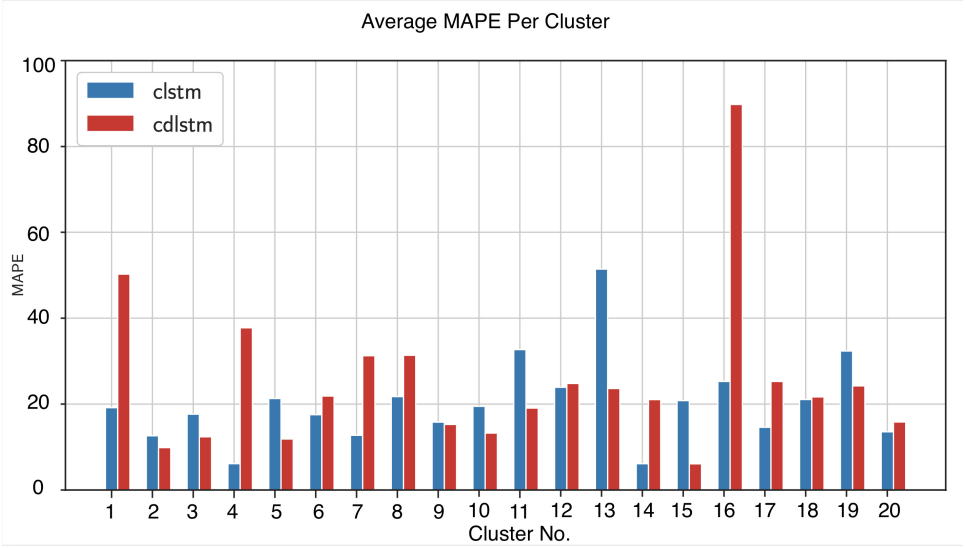
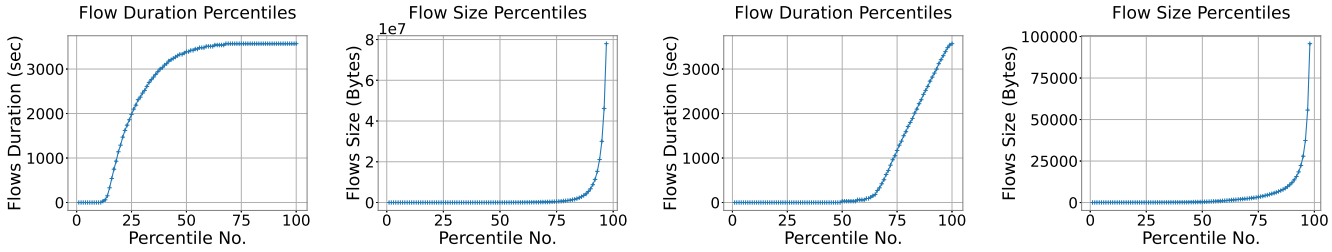


Fig. 9: Average MAPEs for each of the 20 CAIDA flow clusters for the CLSTM and CDLSTM models.



(a) CAIDA flow duration percentiles for prefix size 10.

(b) CAIDA flow size percentiles for prefix size 10.

(c) CAIDA flow duration percentiles for prefix size 20.

(d) CAIDA flow size percentiles for prefix size 20.

Fig. 10: Percentiles for the flow duration and flow size of the CAIDA dataset.

cluster-based LSTM (CLSTM) works better than delta cluster-based LSTM (CDLSTM) in the vast majority of the cases. However, we can observe few cases (e.g. cluster 13) where CDLSTM gave 50% smaller MAPE than CLSTM, something that indicates that using deltas on the time series can reduce the variance of the model.

Finally, in order to better understand the dynamics of the flows in the CAIDA dataset, we plot in Figs. 10(a), 10(b), 10(c) and 10(d) the percentiles of the flow durations (in seconds) and the flow sizes (in bytes) for prefix 10 and 20 respectively. As we can see from the four plots, at larger aggregation scales (i.e. mask size = 10), 80% of the flows appear to be active for more than 750 seconds and only 20% of the flows appear to have a significantly larger size compared to the rest. On the other hand, for smaller aggregation scales (i.e. mask size = 20) 60% of the flows have a duration of 100 seconds or less, and again, only 20% of the flows appear to have a significantly larger size compared to the rest. From the above figures we can infer that AFPDA does not need to rerun very often if we operate at higher aggregation levels, but as we zoom-in, the traffic appears to be more dynamic and more frequent updates will be needed.

B. AFPDA Evaluation

2D-AFPDA Evaluation Using Artificial Data: In order to evaluate the performance of AFPDA in the 2-dimensional prefix plane, we proceed to simulate its evolution under various traffic conditions that capture various probabilistic traffic splits on the prefix trie. 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, and a maximum mask size of `/15`. Specifically, Fig. 11(a) 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 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. 11 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.

1D-AFPDA Evaluation Using The CAIDA Dataset: One optimization we can do in order to speedup the convergence of AFPDA is to leverage potential asymmetries in the cardinalities of the sets of source and destination IPs seen by a switch (depending on the direction, source IPs can be much more than the destination IPs) and break the 2D scan into two phases: 1) 1D scan on the source prefix, and 2) 2D scan on the (reduced) fine-grained prefixes found from step 1) in order to find the destination prefixes that appear to be active. Of course, step 2) above might not be needed depending on the application. In the experiment we present below, we proceed to analyze 1D-AFPDA using the CAIDA dataset since the source IPs are significantly more than the destination IPs. We replay the traffic using the B4 topology [24] where prefixes are randomly assigned to switches and thus prefixes can be measured in various switches using the ILP formulation described in Section V. Every switch has 1024 free TCAM rules that can be used for measurements. As we can see in Fig. 11(c), the total number of large flows found in the CAIDA dataset ranges from 800 to 1500 depending on the threshold θ used, as well as the maximum mask size (i.e. resolution) allowed. The number of flows increases almost linearly with the mask-size and its growth slows down for higher mask values. This shows that there is not significant change in the number of flows found if we increase the depth of the search since most of the large flows are concentrated in individual subnets that have been detected already from higher layers of the prefix trie. In Figs. 11(d), and 11(e) we can see that the measurement duration and the total number of measurements performed also exhibits a linear pattern. However, for smaller thresholds θ, it appears to have an increasing slope that indicates that the algorithm spends more time searching as we approach to the lower levels of the prefix trie. This is expected due to the exponential growth of the prefix trie, however AFPDA does not explore all these states since the network traffic is not distributed across all these prefixes. Finally, as we can see from Fig. 11(f), for each θ value, there is a point after which we can achieve nearly 90% coverage in terms of the total volume of traffic in the network. This shows that AFPDA can indeed be used to find the large flows in the network and keep monitoring them using predictions and measurements interchangeably.

In order to isolate the benefit coming from the PAM algorithm only, we plot the flow coverage gain we get by using predictions for various regression window sizes, and prediction horizons. As we can see from Fig. 12, as well as Eq. 12, and 13, for arbitrary large horizons and small regression windows, we can achieve the highest flow coverage gain compared to an implementation that leverages only the available TCAM in an OpenFlow switch. For example, for $w = 3$ and $T = 6$, we can achieve 2x improvement in terms of the number of flows explored, which can have a big impact in the network visibility.

Finally, in order to investigate if there are any "bottlenecks" in the execution of DeepFlow, we proceed to measure the running times of all its major operations that can be costly, especially the ones related to generating predictions. For this, we have used a 16 core Intel CPU with 32 GB of RAM. The results are shown in Table III, from where we can see the delays for model training, feature extraction, clustering, cluster assignment, prediction generation, and the ILP solver. It is important to note here that DeepFlow operations can be classified in two categories: a) cold start operations, which are needed only during setup and potentially during a periodic retraining (e.g. daily), and b) operations that need to be executed in each epoch. LSTM model training (a cold start operation) can be done in less than 10 minutes for each cluster, but since clusters can be trained in parallel, this process can be completed relatively fast. From the other cold start operations, the data preprocessing for feature extraction can be done in approximately 3 minutes since this task is also parallelizable. In a real system, this operation can also occur on the fly as flows come in. The cluster assignment, and the prediction generation (from a pre-trained model) are pretty fast operations and even in our Python prototype, they took approximately 0.1 seconds to complete. Finally, the ILP solver terminates in less than 2.81 seconds since the number of switches and rules to install are relatively small. In addition, since the progression of DeepFlow occurs in a Round-Robin fashion, we can easily cache past solutions and avoid solving the ILP in each epoch, in which case ILP also can be considered a cold start operation. From the above we can conclude that building a measurement system that leverages predictions can scale well since the operations that need to occur in real time have very small latency or can be precalculated.

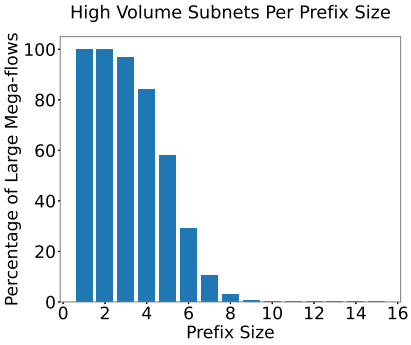
TABLE III: DeepFlow Latency Breakdown

DeepFlow Stage	**Execution Time**
LSTM Model Training (Cold Start)	5-550 secs
Feature Extraction (Cold Start)	186 secs
Clustering (Cold Start)	274 secs
Cluster Assignment	0.1 secs
Generate Prediction	0.1 secs
ILP solver	2.81 secs

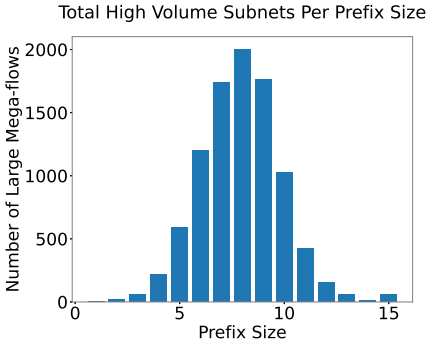
C. Discussion

Comparison With Top-K Methods: From the analysis presented above, we can see that DeepFlow can be seen as a generalization of top-K or Heavy Hitter (HH) detection methods since it goes beyond what these methods can achieve both in the spatial and the temporal dimension. In other words, a network administrator dashboard using DeepFlow can provide time-series continuously in regards to the most important flows that are more than the top-K flows. However, a top-K or HH approach can only provide up to K largest flows where K is limited by the available TCAM.

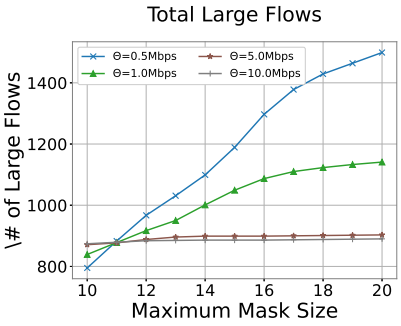
Measurement During Dynamic Rule Updates: Although in this paper we focus on proactive SDN deployments where all the flows have a matching prefix in the TCAM, it is possible that rule updates might be requested during a measurement interval, e.g. when the controller application tries to change the state of the network during Traffic Engineering (TE). In practice, there are some scenarios where the controller can choose to wait until the end of the measurement epoch before updating the TCAM (since DeepFlow is essentially a controller application as well) if this does not incur data loses or other negative consequences (e.g. TE decisions can



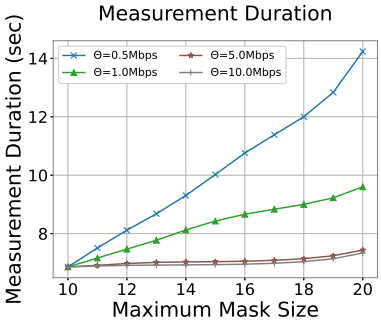
(a) The percentage of high volume prefixes for each source and destination IP mask size, as determined by AFPDA using random traffic splitting.



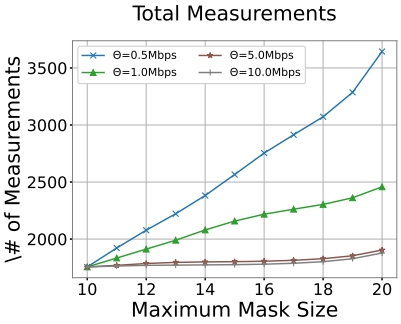
(b) The total number of high volume prefixes for each source and destination IP mask size, as determined by AFPDA using random traffic splitting.



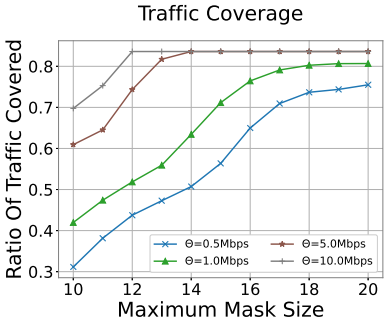
(c) Number of large flows found for various θ and d_s.



(d) Measurement duration for various θ and d_s.



(e) Total AFPDA measurements for various mask sizes for epoch = 15 sec.



(f) Total coverage (i.e. percentage of the total volume) of the significant flows found for various mask sizes for epoch = 30 sec.

Fig. 11: AFPDA Statistics for a 2D parsing of an artificial dataset, as well as a 1D parsing of the CAIDA dataset, in order to assess where significant flows are located.

be applied during multiples of the epoch size). However, it might be imperative that the controller enforces rule updates during the measurement epoch. In such a case, DeepFlow will operate as follows. a) If a new rule needs to be added and there is no free TCAM rule, then two children rules can be merged into one and free up one rule. This can be done with 2 DEL and 1 + 1 ADD rules. However, it can also be done with 1 MOD (i.e. update) rule only, as long as the rules to be merged are of the form ("parent prefix", and "offloaded children prefix with higher priority"). For example, in Fig. 2, we can update the rule `11` to correspond to the new rule of interest, as long as the existing rules are `1*` and `11` instead of `10`, and `11`. In a similar fashion, if a rule needs to be deleted or modified, the controller can instead apply a MOD operation and replace the target rule with a prefix that is due to be measured, or the actual target rule that needs to be modified, respectively. In order to synchronize the data across switches and achieve a network-wide update, DeepFlow will execute a simple heuristic and in case of k rule additions, it will find the smallest k flows that can be merged with their parent throughout the switches in the paths of the k flows, and in case of k rule deletions, it will find the k rules with the smallest remaining TTLs that are due for measurement again and can be installed in the switches in the path of the flows to be deleted. This way, the flow coverage will continue to be the maximum possible, with the only difference being that some flow predictions will occur for smaller or larger time windows without affecting our target performance metrics.

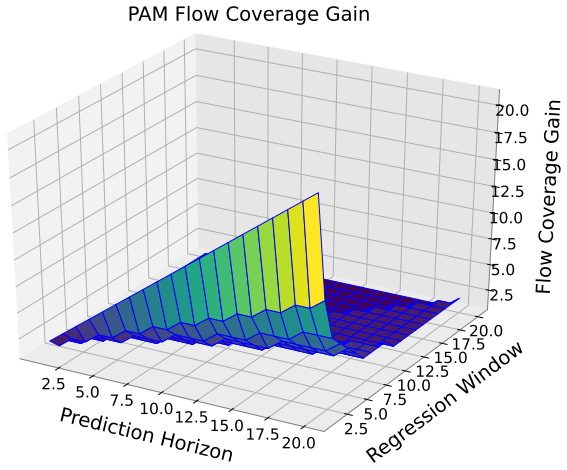


Fig. 12: Flow coverage gain achieved due to predictions for various regression window sizes and prediction horizons.

VIII. 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 to further reduce the number of exact flow measurements needed.

REFERENCES

[1] Augustin Soule, Kavé Salamatian, and Nina Taft. 2005. "Combining filtering and statistical methods for anomaly detection". In Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement (IMC '05). USENIX Association, Berkeley, CA, USA, 31-31.

[2] 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.

[3] 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, Tokyo Japan.

[4] Andrew R. Curtis, Jeffrey C. Mogul, Jean Tourrilhes, Praveen Yalagandula, Puneet Sharma, and Sujata Banerjee. 2011. "DevoFlow: scaling flow management for high-performance networks". SIGCOMM Comput. Commun. Rev. 41, 4 (August 2011), 254-265.

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

[6] 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.

[7] Tune, Paul, and Matthew Roughan. "Internet traffic matrices: A primer." Recent Advances in Networking 1 (2013).

[8] Brandon Heller, Srini Seetharaman, Priya Mahadevan, Yiannis Yiakoumis, Puneet Sharma, Sujata Banerjee, and Nick McKeown. 2010. ElasticTree: saving energy in data center networks. In Proceedings of the 7th USENIX conference on Networked systems design and implementation (NSDI'10). USENIX Association, USA, 17.

[9] Liang Zhou, Laxmi N. Bhuyan, and K. K. Ramakrishnan. 2019. DREAM: DistRibuted Energy-Aware traffic Management for Data Center Networks. In Proceedings of the Tenth ACM International Conference on Future Energy Systems (e-Energy '19). Association for Computing Machinery, New York, NY, USA, 273-284.

[10] L. Chiaraviglio, M. Mellia and F. Neri, "Minimizing ISP Network Energy Cost: Formulation and Solutions," in IEEE/ACM Transactions on Networking, vol. 20, no. 2, pp. 463-476, April 2012.

[11] Will Fisher, Martin Suchara, and Jennifer Rexford. 2010. Greening backbone networks: reducing energy consumption by shutting off cables in bundled links. In Proceedings of the first ACM SIGCOMM workshop on Green networking (Green Networking '10). Association for Computing Machinery, New York, NY, USA, 29-34.

[12] OpenFlow Specifications Version 1.5.1: https://www.opennetworking.org/software-defined-standards/specifications/

[13] 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.

[14] 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.

[15] J. Kucera, D. A. Popescu, G. Antichi, J. Korenek, and A. W. Moore, "Seek and Push: Detecting Large Traffic Aggregates in the Dataplane.", CoRR, abs/1805.05993, May 2018.

[16] 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.

[17] Moshref, Masoud, et al. "SCREAM: Sketch Resource Allocation for Software-defined Measurement." CoNEXT, Heidelberg, Germany (2015).

[18] 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 Conference. ACM, 2016.

[19] 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 (SIGCOMM '14). ACM, New York, NY, USA, 419-430.

[20] 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.

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

[22] 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.

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

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

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

[26] Aggelos Lazaris and Viktor K. Prasanna. 2017. DeepFlow: a deep learning framework for software-defined measurement. In Proceedings of the 2nd Workshop on Cloud-Assisted Networking (CAN '17). ACM, New York, NY, USA, 43-48.

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

[28] Zhou, Haifeng, et al. "Traffic matrix estimation: A neural network approach with extended input and expectation maximization iteration." Journal of Network and Computer Applications 60 (2016): 220-232.

[29] Li, Yuliang, et al. "FlowRadar: a better NetFlow for data centers." 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16). 2016.

[30] Yu, Ye, Chen Qian, and Xin Li. "Distributed and collaborative traffic monitoring in software defined networks." Proceedings of the third workshop on Hot topics in software defined networking. ACM, 2014.

[31] Malboubi, Mehdi, et al. "Intelligent sdn based traffic (de) aggregation and measurement paradigm (istamp)." IEEE INFOCOM 2014-IEEE Conference on Computer Communications. IEEE, 2014.

[32] A. Lazaris, D. Tahara, X. Huang, L.E. Li, A. Voellmy, Y.R. Yang and M. Yu, "Jive: Performance Driven Abstraction and Optimization for SDN", Open Networking Summit (ONS), Santa Clara, CA, April 2014.

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

[34] 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.

[35] 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.

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

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

[38] Bob Lantz, Brandon Heller, and Nick McKeown. 2010. A network in a laptop: rapid prototyping for software-defined networks. In Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks (Hotnets-IX). ACM, New York, NY, USA, , Article 19 , 6 pages. Download Link: http://mininet.org/

[39] A. Lazaris and V. K. Prasanna, "An LSTM Framework For Modeling Network Traffic," 2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), Arlington, VA, USA, 2019, pp. 19-24.

[40] Open Virtual Switch (OpenVSwitch): http://openvswitch.org/

[41] Jiang, Dingde, and Guangmin Hu. "Large-scale IP traffic matrix estimation based on the recurrent multilayer perceptron network." 2008 IEEE International Conference on Communications. IEEE, 2008.

[42] Ryu, SDN Controller in Python: http://osrg.github.io/ryu/

[43] Paul Goransson, Chuck Black, "Software Defined Networks, A Comprehensive Approach", Morgan Kaufmann, 2014.

[44] 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.

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

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

[47] A. Lazaris, D. Tahara, X. Huang, E. Li, A. Voellmy, Y. R. Yang, and M. Yu. “Tango: Simplifying SDN Control with Automatic Switch Property Inference, Abstraction, and Optimization”. In Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies (CoNEXT ’14). ACM, New York, NY, USA, 199-212.
[48] A. Yassine, H. Rahimi and S. Shirmohammadi, ”Software defined network traffic measurement: Current trends and challenges,” in IEEE Instrumentation & Measurement Magazine, vol. 18, no. 2, pp. 42-50, April 2015.
[49] Y. Zhang, ”An adaptive flow counting method for anomaly detection in SDN,” in Proceedings of the ninth ACM CoNEXT’13, 2013, pp. 25–30.
[50] R. Xu and D. Wunsch Ii, ”Survey of Clustering Algorithms,” IEEE Trans. NEURAL NETWORKS, vol. 16, no. 3, 2005.
[51] N. Ramakrishnan and T. Soni, ”Network Traffic Prediction Using Recurrent Neural Networks,” 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), Orlando, FL, 2018, pp. 187-193.
[52] T. Saydam and T. Magedanz, ”From networks and network management into service and service management,” J. Netw. Syst. Manag., vol. 4, no. 4, pp. 345–348, Dec. 1996.
[53] D. Jiang, Y. Wang, Z. Lv, W. Wang and H. Wang, ”An Energy-Efficient Networking Approach in Cloud Services for IIoT Networks,” in IEEE Journal on Selected Areas in Communications, vol. 38, no. 5, pp. 928-941, May 2020.
[54] D. Jiang, W. Wang, L. Shi and H. Song, ”A Compressive Sensing-Based Approach to End-to-End Network Traffic Reconstruction,” in IEEE Transactions on Network Science and Engineering, vol. 7, no. 1, pp. 507-519, 1 Jan.-March 2020.
[55] D. Jiang, L. Huo and H. Song, ”Rethinking Behaviors and Activities of Base Stations in Mobile Cellular Networks Based on Big Data Analysis,” in IEEE Transactions on Network Science and Engineering, vol. 7, no. 1, pp. 80-90, 1 Jan.-March 2020.
[56] D. Jiang, Y. Wang, Z. Lv, S. Qi and S. Singh, ”Big Data Analysis Based Network Behavior Insight of Cellular Networks for Industry 4.0 Applications,” in IEEE Transactions on Industrial Informatics, vol. 16, no. 2, pp. 1310-1320, Feb. 2020.
[57] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. 2018. Elastic sketch: adaptive and fast network-wide measurements. In Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM ’18). Association for Computing Machinery, New York, NY, USA, 561-575.
[58] Qun Huang, Xin Jin, Patrick P. C. Lee, Runhui Li, Lu Tang, Yi-Chao Chen, and Gong Zhang. 2017. SketchVisor: Robust Network Measurement for Software Packet Processing. In Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM ’17). Association for Computing Machinery, New York, NY, USA, 113-126.
[59] M. Dayarathna, Y. Wen and R. Fan, ”Data Center Energy Consumption Modeling: A Survey,” in IEEE Communications Surveys & Tutorials, vol. 18, no. 1, pp. 732-794, 2016.

Aggelos Lazaris received the BS and MS degree in Electronics and Computer Engineering from the Technical University of Crete, Greece, and the PhD degree in Electrical Engineering from the University of Southern California (USC), Los Angeles. His research interests span the area of Computer Communication Networks with emphasis on Traffic Measurement and Analysis, Software-Defined Networks, and the application of Machine Learning in order to built smarter and more efficient networks. He is currently the Chief Data Scientist at a cybersecurity startup developing real-time measurement-based solutions to combat botnets.

Viktor K. Prasanna received the BS degree in Electronics Engineering from the Bangalore University, the MS degree from the School of Automation, Indian Institute of Science, and the PhD degree in computer science from Pennsylvania State University. He is Charles Lee Powell Chair in Engineering and is Professor of Electrical and Computer Engineering and Professor of Computer Science with the University of Southern California (USC). He is the director of the Center for Energy Informatics (CEI) and the USC-Infosys Center for Advanced Software Technologies (CAST) at USC. He is an associate member of the Center for Applied Mathematical Sciences (CAMS) and leads the Parallel Computing/FPGA (fpga.usc.edu) and Data Science (dslab.usc.edu) Labs. The Parallel Computing/FPGA Lab is focused on innovative applications of FPGAs for accelerating computations in diverse areas while the Data Science Lab is exploring novel applications of machine learning in complex engineered systems. He was an associate director of the USC Chevron Center of Excellence for Research and Academic Training on Interactive Smart Oilfield Technologies (Cisoft). His research interests include High-Performance Computing, Parallel and Distributed Systems, Reconfigurable Computing, Cloud Computing, and Smart Energy Systems. He served as the editor-in-chief of the IEEE Transactions on Computers during 2003-06. Currently, he is the editor-in-chief of the Journal of Parallel and Distributed Computing. He was the founding chair of the IEEE Computer Society Technical Committee on Parallel Processing. He is the steering chair of the IEEE International Parallel and Distributed Processing Symposium (IPDPS) and the IEEE International Conference on High Performance Computing (HiPC). He received the W. Wallace McDowell Award from the IEEE Computer Society in 2015 for his contributions to reconfigurable computing. He received an Outstanding Engineering Alumnus Award from the Pennsylvania State University in 2009 and the Distinguished Alumnus Award from the University Visveswaraya College of Engineering (UVCE) of Bangalore University in 2017. He was appointed Distinguished Professor, Beihang University (BUAA), Beijing, PRC, in 2018. He received a 2019 Distinguished Alumnus Award from the Indian Institute of Science (IISc). His work on regular expression matching received one of the most significant papers in FCCM during its first 20 years award in 2013. He is a fellow of the IEEE, the ACM, and the American Association for Advancement of Science (AAAS).