Heavy Hitter Detection and Identification in Software Defined Networking

Liang Yang, Bryan Ng, Winston K.G. Seah
School of Engineering and Computer Science, Victoria University of Wellington
P.O. Box 600, Wellington 6140, New Zealand
{ liang.yang, bryan.ng, winston.seah }@ecs.vuw.ac.nz

***Abstract*—In a large network, it is often important to be able to detect high-volume traffic in near real-time. Existing work on the detection and identification of such high volume traffic (so-called *heavy hitters*) is typically delegated to individual nodes and often relies on deep packet inspection and/or packet sampling. However, these techniques have well known limitations in terms of its ability to scale with network size. Inspired by the capabilities of Software Defined Networking (SDN), we explore a novel heavy hitter detection solution based on understanding connections between traffic statistics and OpenFlow rules. Our approach relies on mining traffic statistics (e.g. port bitrate) and forwarding table entry (FTE) to improve heavy hitter detection. The rationale behind this approach are (i) the information is readily available with minimal overheads, thus it scales better with increasing network size; and (ii) the FTEs and traffic statistics provide different vantage for detection and identification of heavy hitters. We evaluate the effectiveness and accuracy of our proposed heavy hitter detection algorithm on a test bed as a proof-of-concept. The test results show that our approach to heavy hitter detection simultaneously achieves considerable accuracy and good scalability.**

***Index Terms*—Heavy Hitter, SDN, FTE**

I. Introduction

The phenomena of small number of flows carrying majority of bytes has been studied in many incarnations in computer networking [1], [2]. A simple way to define these "important" traffic are the terms *elephant flow* or *heavy hitter* (HH). They can be identified by the common characteristics in the packets of these flows, for example, the source IP prefix or destination MAC address. The detection and identification of these flows are critical for Quality of Service provisioning and traffic load balancing.

To find these heavy hitters in a network, two general approaches have been widely used. One approach is based on statistical sampling which is used in NetFlow [3] and sFlow [4]. In these solutions, sampled packets are sent to a central server that aggregates count values and identifies the heaviest flows. Another approach is based on a specialized data structures called a "sketch". The *sketch* summarizes the ingress traffic and this summary is examined to get the approximate amount of the packets belonging to the same flow in several passes. These so-called sketch-based methods have significant advantages in situations where storing the entire data set would be prohibitively costly.

Most of these existing works focus on the detection of heavy hitters on individual switches. However, in a wide range of applications such as network monitoring, security and load balancing [5], finding heavy hitters across multiple switches is a problem of increasing significance. To address this problem, one straightforward solution is for all switches to send their sampled packets or measurements to a central server which is responsible for detecting and identifying the heavy hitters. There are two major drawbacks of this naïve solution. Firstly, the communication overheads between switches and servers is high because each switch has to send header or even payload of each sampled packet to the server for further processing. Secondly, the time to screen, inspect and count all incoming sampled packets is significant. A medium-sized network has hundreds of switching. Each of these nodes has up to 128 Ethernet ports operating at the typical speeds of 10Gbps, 40Gbps or more. It is almost impossible to process all samples in real-time, which is not acceptable for some applications, for example, load balancing, firewall, etc.

A new networking paradigm, *software defined networking* (SDN), separates the data plane from the control plane, turning network switches in the data plane into simple packet forwarding devices. A logically centralized software program controls the behavior of these switches and by extension controls the entire network. In a software-defined network, a controller is able to retrieve the switch statistics with varying granularity, which opens up opportunities to develop better approaches to detect and identify HHs. To make the HH detection feasible in practice, careful consideration must be given to the communication overheads as well as the processing time for sampled data.

In this paper, we propose a lightweight heavy hitter detection and identification solution that incurs lower communication and processing overheads compared to traditional sampling-based technology. It is a software solution based on multiple passes of retrieving and analyzing network traffic statistics. A controller can retrieve statistics from the switches at three levels [6]: (i) OpenFlow table, (ii) aggregate flow, and (iii) individual flow. By analyzing these coarse-to-fine statistical information, the suspected HHs are gradually narrowed down and finally identified. We demonstrate that our proposed solution meets the functional tests for HH detection via testbed experiments and show that our solution is scalable and efficient via analysis.

II. Related work

Sampling-based measurements of network traffic such as NetFlow and sFlow have been widely used in heavy hitter detection. NetFlow [3] is a feature on Cisco routers for collecting IP network traffic with predefined rules. sFlow [4], short for "sampled flow", provides a means for collecting information of truncated packets, together with the interface statistics. Unlike NetFlow, sFlow offers greater scalability and reports layer 2 to layer 7 information on network traffic in detail [7] but it consumes more resources (e.g., CPU, memory, and bandwidth). A high sampling rate generates too much information (costly to store), while a lower sampling rate may result in HH flows going undetected.

Compared with sampling-based measurement, sketch-based detection can process millions of streams in a short time and with low overheads. It is a probabilistic summary of data streams within a compact data structure. The approach builds forecast models on top of sketches which represent the past traffic patterns. Sketch-based approach adopts a unique hash function and associates multi-dimensional tables to data streams for storing summarized data, which requires customized designs of existing switch ASIC (application–specific integrated circuit). This is why most of the existing works are only verified by simulation and implemented on field-programmable gate array (FPGA) [8], [9].

There are two major problems with the existing approaches. The first problem is excessive overheads. Sampling-based technique is extremely resource intensive because of the packet inspection process while sketch-based solution is highly dependent on proprietary hardware. Another problem is that they lack the capability to detect the multi-dimensional heavy hitters (MHH) because the packet header fields (e.g., MAC address, IP address and port numbers) must be defined in advance for accurate detection. However, it is difficult to identify which header fields to monitor because MHHs are aggregate flows that have headers permuted from its constituent flows. These have been shown to originate from virtual machines (VMs) in which traffic is aggregated from various applications. Without prior knowledge about the patterns of such traffic, all possible combinations of aggregates must be examined to detect the heavy hitters.

In the study of multi-dimensional traffic, one notable type of heavy hitter is called hierarchical heavy hitter (HHH) [10], [11]. Autofocus is a good attempt to automatically characterize HHH traffic based on IP addresses [12]. However, the research on unstructured heavy hitters has not been fully explored. There are numerous examples that show that aggregate traffic does not follow any known pattern [13]. To address these problems, this paper proposes a network-wide and traffic pattern unaware heavy hitter detection solution.

III. Design of the HH detection framework

We propose a generic SDN application framework that retrieves and processes the network's information in a hierarchical and staged manner to reduce the: (i) computation overheads on the controller and (ii) communication overheads between switches and controller. In comparison to a sFlow or NetFlow server in a conventional network, a centralized controller in SDN has a broader view of the devices under its control. A SDN controller can leverage real-time status and statistics from these devices to make better decisions about how to deploy or optimize them for HH detection. However, it is infeasible to request every switch to report all flow statistics due to the enormous communication overheads in a network and the excessive processing to be done by the controller.

Besides the switch-level and table-level (coarse-grained) statistics, a controller also deals with more fine-grained statistics such as the status of ports, meter, CPU queue, etc. [6]. The key challenge of designing a SDN-based HH detection solution is to strike a careful balance between generality (supporting a wide variety of statistics) and efficiency (low communication cost and computation overheads) [9].

Two types of inputs are used in our HH detection framework: traffic statistics and OpenFlow tables. Traffic statistics enable the controller to understand the network runtime status while OpenFlow tables define the underlying paths across the network. To avoid input deluge to the controller, we use a multi-stage coarse to fine approach to stagger the inputs for HH detection. A controller begins with the coarse-grained information such as table or port level statistics for the initial investigation, and then selects the fine-grained flow-level information for further processing. This two-stage input processing design allows the controller to focus on a small set of information in each stage, thus reducing communication and computation costs.

The rationale behind the multi-stage approach is that aggregate traffic retains some characteristics of each individual flow, especially the most significant one. In fact, all traffic can be considered as aggregate flows with different granularities. The traffic in a single Ethernet card is an aggregation of flows from all applications on that server; the traffic in a physical switch port is an aggregation of flows from several servers or switches. Early traffic models were derived from telecommunications networks and generally operated under the assumption that aggregate traffic from a large number of sources smooths out bursts [14]. However, today's networks are designed to scale-out rather than to scale-up, which makes these aggregate traffic traceable and their hidden HHs detectable [15].

A. Modules of the HH detection framework

The HH detection framework consists of a detection module and an identification module. The detection module determines if a HH is likely to be present in a flow while the identification module positively establishes the HH. The worklow for the HH detection and identification framework in Fig. 1 depicts two-stages and each stage is instantiated as a module.

In Stage 1 (detection module) of Fig. 1, the SDN controller requests all switches to report their coarse level information such as OpenFlow table status or port packet counters. After the initial analysis, the controller filters out the messages in which it has no interest. Then the controller requests switches to send the finer aggregate statistics (e.g. per-flow level)

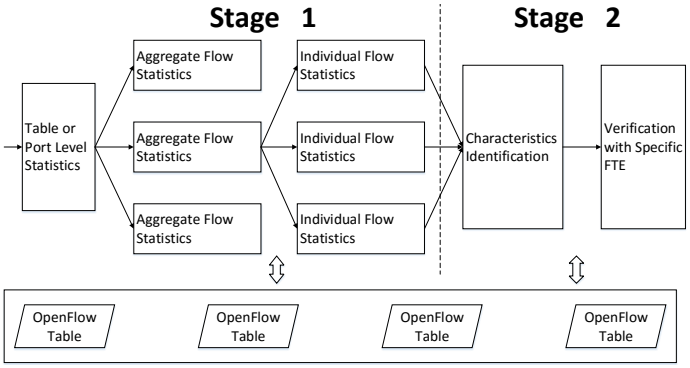

Fig. 1. The proposed HH detection framework as a two-stage process. Each stage is instantiated as a module.

for further processing. Since FTEs determine the forwarding behavior of all the underlying flows, the characteristics of these flows can be reversely inferred from its matched FTEs. In Stage 2 (identification module) of Fig. 1, a new FTE in which the match fields are constructed based on the characteristics of the detected heavy hitter is created and installed into the switches along the path where HH are detected. By monitoring the traffic statistics of these new added FTEs, we can verify whether a detected flow is indeed a HH or just an aggregation of multiple mice flows.

B. Detection Module

The purpose of detection module is to distinguish the elephant flows from each traffic data stream and group them based on certain measure of similarity. This is achieved through processing the information from switches such as outgoing counters or bytes as a time series. The time series model [16] is adopted to describe the traffic from a physical port in a switch. It is a sequence of traffic data points which consist of successive measurements made over a constant time interval.

As illustrated in Fig. 2, the detection module consists of four procedures: smoothing, thresholding, windowing, and correlating. Smoothing removes the spikes in the time series, thresholding determines the interval in which an elephant flow occurs, windowing isolates these elephant flows after applying a threshold. The effect of these three procedures on a raw traffic stream are illustrated in Fig. 3. Finally, these windowed datasets are grouped based on their correlations and each group represents a set of elephant flows with high similarities, assumed to be the same elephant flow pending further diagnostic. Before we detail each procedure in the detection module we first define a heavy hitter:

Definition 1 - Heavy Hitter.

$\mathscr{L}$ is a set of points indexed by k for a single flow: $\mathscr{L} = \{\alpha_k\} : \forall k \in \mathbb{Z}_0^+$. Let L denote the minimum period for observing a HH, and let I denote maximum period in which all sampled rates are less than the threshold T; a HH $\mathscr{H}$ is a subset of $\mathscr{L}$, $\mathscr{H} = \{\alpha_k\}_{k=i}^{j} : (j-i+1) \geq L, (\nexists m, n : i \leq m \leq n \leq j, (n-m+1) \geq I, \{\alpha_k\}_{k=m}^{n} < T)$.

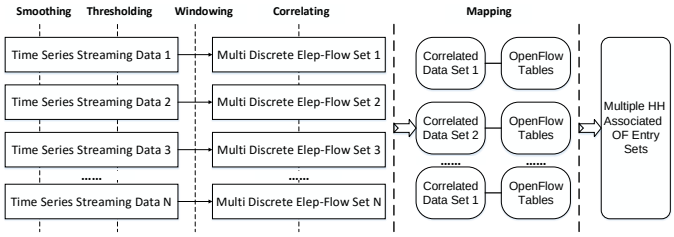

Fig. 2. The workflow of four procedures within the detection module.

1) Smoothing: Smoothing is the procedure that removes "noise" from a data stream. For an aggregate traffic dataset, the elephant flows are considered as the desired signal and the mice flows as noise. We use the k-nearest neighbor algorithm to filter out spikes and noise from the time-series of an aggregate flow and this procedure is outlined in Algorithm 1.

Algorithm 1 Minimum k-nearest neighbors algorithm

```
procedure SMOOTHING(k)
    R ← input: Two dimensional array of raw traffic data
    S ← output: Two dimensional array of smoothing data
    M ← Absolute maximum value
    for i = 0, i < len(R)/k, i++ do
        for j = 0, j < k, j++ do
            if R_{i*k+j} < M then
                M ← R_{i*k+j}
            end if
        end for
        S_i ← M
    end for
end procedure
```

2) Thresholding: After smoothing the incoming traffic dataset S_i (where i is the index of that dataset), a threshold is applied to S_i. All the flows above a certain threshold value T are considered as potential elephant flows. The thresholding process is expressed in the following equation:

$$T_i = \begin{cases} S_i - T & if \quad S_i \geq T \\ 0 & if \quad S_i < T \end{cases} \tag{1}$$

3) Windowing: In practice, we cannot observe T_i on the interval $(-\infty, +\infty)$. One way of overcoming this problem is to split every traffic stream into multiple fixed time interval datasets. According to Definition1, an elephant flow or HH is a flow in which the longest continuous period below the given threshold is no more than t with duration no less than l. For the post-threshold dataset T_i, the problem of identifying potential HH is simplified to finding all datasets with the following three attributes: i) the length of each dataset is no less than l; ii) the number of consecutive zeros is smaller than t; and iii) begin and end with non-zero value. This windowing procedure is shown in Algorithm 2.

4) Correlating: If a HH traverses multiple switches, it can be observed from the statistics of the FTEs in these switches. The procedure of *correlating* is used to find the HHs

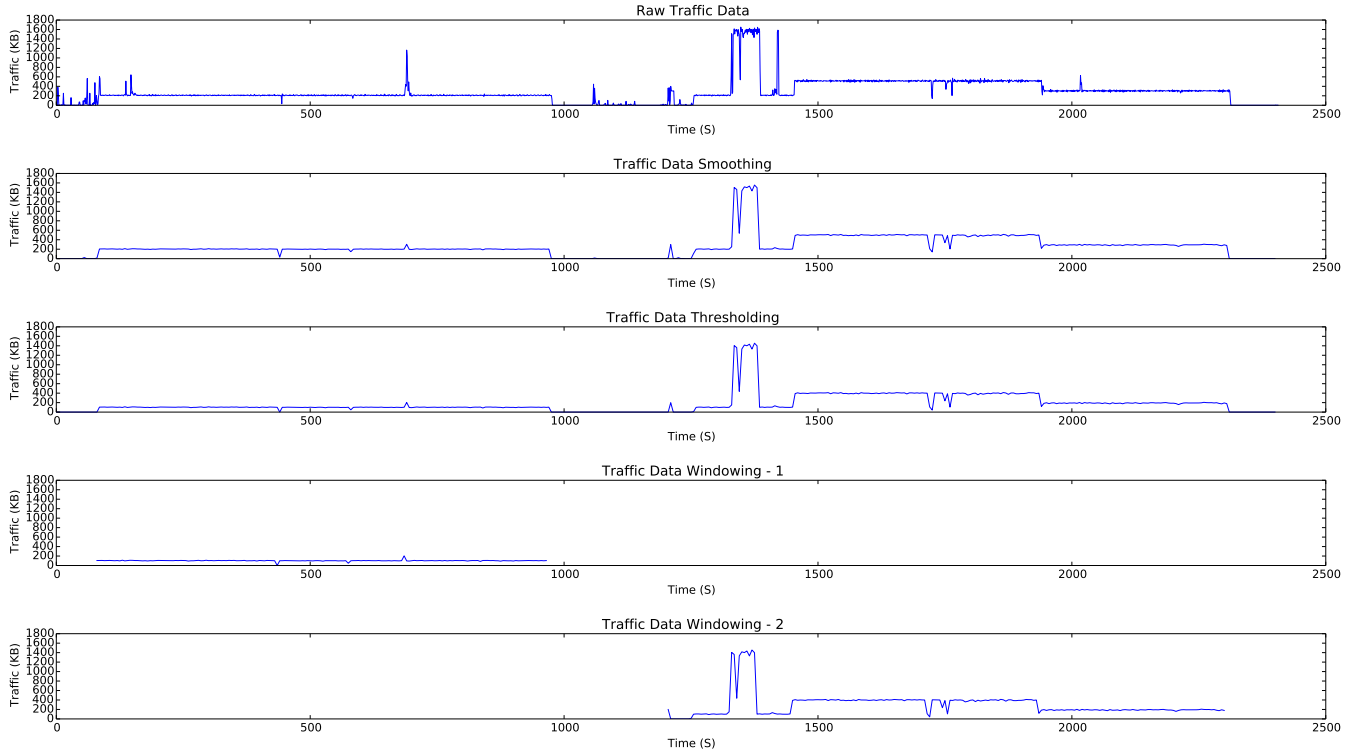

Fig. 3. Smoothing-Thresholding-Windowing

Algorithm 2 Windowing

```
1:  procedure WINDOWING
2:      T ← input: Two dimensional array thresholding data
3:      W ← output: Two dimensional array windowing data
4:      Z ← Array to save consecutive 0 in T
5:      t ← Treat as same flow with consecutive 0 less than t
6:      l ← The minimum length of a window
7:      n ← 0, Number of windows
8:      for i = 0, i < len(T), i++ do
9:          if T_i = 0 then
10:             if (len(W_n) ≥ l) & (len(Z)) ≥ t) then
11:                 n ← n+1
12:             end if
13:             Append T_i to Z
14:         else
15:             if T_i-1 = 0 &
                   (len(W_n) > 0) & (len(Z) < t) then
16:                 Append Z to W_n
17:             end if
18:             Z ← 0
19:             Append T_i to W_n
20:         end if
21:     end for
22: end procedure
```

belonging to the same flow based on their similarities. There are two major categories of similarities: bias oriented and variance oriented. If bias outweighs variance as an indicator of similarity, the bias oriented similarity is selected. In this case, two datasets with exact fluctuations will be considered as dissimilar if the difference of their mean values exceed a certain threshold.

On the contrary, the variance oriented similarity values the weights of deviations more than the difference of their mean values. For an aggregate traffic stream, the variance of an elephant flow is not easily identified because it is obscured by other mice flows. In our solution, the bias oriented similarity will be applied to find the possibility whether two aggregate flows hide the same elephant flow.

The similarity between two time series is typically measured with Euclidean distance. The Euclidean distance is suitable for our purpose because there are no offset and acceleration-deceleration along the time dimension to be taken into consideration after the time series passes through the windowing procedure. However, the problem of storing large amounts of data persists because of the following three factors prevalent in large networks: (i) over hundreds of time series data streams; (ii) large volume per dataset (since each unit of sampling traffic data takes one byte in memory, the total size of a single 60-minute dataset will be as high as $3.6MB$); and (iii) fast response time for detecting HH.

In this paper, Adaptive Piecewise Constant Approximation (APCA) [17] is adopted to represent the original time series data for similarity search (see Appendix A for comparison of APCA and other related techniques). The APCA representa-

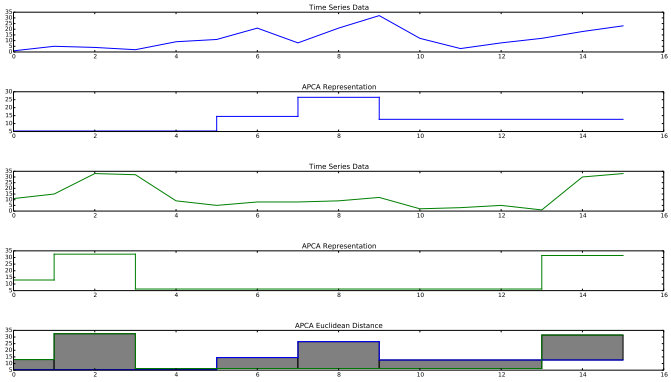

Fig. 4. APCA Euclidean distance

tion of a time series $C = \{c_1, \ldots, c_n\}$ can be expressed as:

$$C = \{\langle cv_1, cr_1\rangle, \ldots, \langle cv_M, cr_M\rangle\}, cr_0 = 0 \qquad (2)$$

where cv_i is the mean value of datapoints in the *i*-th segment and cr_i is the right endpoint of the *i*-th segment. More detailed discussion on reducing data dimensionality is given in Appendix A.

C. Detecting similarity

The approximate Euclidean distance between a measured dataset and a time series in APCA representation was derived in [17]. Considering the APCA representation C as a reconstructed time series, the Euclidean distance between a time series Q and C is defined as:

$$D_{AE}(Q, C) = \sqrt{\sum_{i=1}^{M} \sum_{i=1}^{cr_1 - cr_{i=1}} (cv_i - q_{k+cr_{i=1}})^2}. \qquad (3)$$

To further reduce the computation time, a coarser approximation of Euclidean distance between two query time series $A = \{\langle av_1, ar_1\rangle, \ldots, \langle av_m, cr_m\rangle\}$ and $B = \{\langle bv_1, br_1\rangle, \ldots, \langle bv_n, br_n\rangle\}$, $D_{CE}(A, B)$, is used in our solution. This distance is calculated directly based on their respective APCA representation and defined as:

$$D_{CE}(A, B) = \sqrt{\sum_{i=1}^{m} \sum_{i=1}^{n} (av_i - bv_j)^2 |ar_i - br_j|} \qquad (4)$$

where $(ar_{i\text{-}1} \leq br_j \leq ar_i)$ or $(br_{j\text{-}1} \leq ar_i \leq br_j)$.

Although the distance $D_{CE}(A, B)$ is not a lower bound measure, it is still very useful for approximate search in a large dataset. The procedure to compute the distance between APCA representations is described in Algorithm 3.

Fig. 4 illustrates the APCA distance computation of two datasets, each of them consists of 16 elements. APCA distance will be used to determine the similarity of two HHs. If their distance is below a given threshold which is also called similarity threshold (ST) in this paper, they will be considered as the same flow.

Algorithm 3 Euclidean distance of APCA Representation

procedure APCA_DISTANCE
  A, B ← input: Two APCA datasets
  $E, AR, BR \leftarrow 0$ // temporary vectors
  $AV \leftarrow$ all av_i in A; $BV \leftarrow$ all bv_i in B
  D ← output: Euclidean distance of A and B
  for ar_i in A **do**
    Append ar_i to E and AR
  end for
  for br_i in B **do**
    Append br_i to E and BR
  end for
  Sort, Insert 0 at $E[0]$ and delete $E[len(E) - 1]$
  a_value ← $AV[0]$; *b_value* ← $BV[0]$
  for $i = 0$, $i < len(E) - 1$, i++ **do**
    if E_i **in** AR **then**
      index ← the index of item E_i in *AR*
      a_value ← AV_{index}
    else
      if E_i **in** BR **then**
        index ← the index of item E_i in *BR*
        b_value ← BV_{index}
      end if
    end if
    D+=$(|a\_value - b\_value|) * (E_{i+1} - E_i)$
  end for
end procedure

Match	Priority	Counters	Instructions	Timeouts	...

Fig. 5. A single flow table entry

D. HH Detection via match set analysis

A typical OpenFlow FTE is composed of *match set*, *actions(instructions)* and *priority*, this is shown in Fig. 5. Usually, a match set is composed of multiple fields which identify an individual flow or a set of flows. In the medium access control (MAC) layer, it includes destination MAC (DMAC) and source MAC (SMAC) addresses, while in the network layer, it is a 5-tuple consisting of source IP address (SIP), port number, destination IP address (DIP), port number and the protocol in use. In most cases, actions contain the output ports, which helps relate traffic statistics and their corresponding FTEs. Priority indicates matching precedence of a flow entry which means the matching entry with highest priority will determine how a packet is forwarded.

The relations that can be in place among match sets and instruction sets were first analyzed in [18]. Based on the potential relation combinations, they define five FTE operator types: *Disjoint*, *Exactly matching*, *Subset*, *Superset*, and *Correlated*. A brief overview of these operators is given in Appendix B.

A HH that matches multiple FTEs, can be expressed by a function of match fields and set operators (explained in Appendix B). This approach of using one or more match fields to identify a HH is also called *HH match set*. Suppose there are

TABLE I

OpenFlow Sample Table A

Index	FTE
1	Match: M_1^A, Action: Drop
2	Match: M_2^A, Action: output 21
3	Match: M_3^A, Action: output 2
i	...
$i+1$	Match: M_{i+1}^A, Action: output 2,21,24
$i+2$	Match: M_{i+2}^A, Action: output: ALL
$i+3$	Match: ALL, Action: output: CONTROLLER

TABLE II

OpenFlow Sample Table B

Index	FTE
1	Match: M_1^B, Action: output 4
2	Match: M_2^B, Action: output Drop
3	Match: M_3^B, Action: output 4, 18,22
j	...
$j+1$	Match: M_{j+1}^B, Action: output 4,20
$j+2$	Match: M_{j+2}^B, Action: output: ANY
$j+3$	Match: ALL, Action: output: CONTROLLER

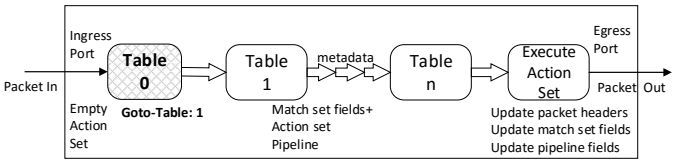

Fig. 6. Table "0" Reservation for HH identification in OpenFlow

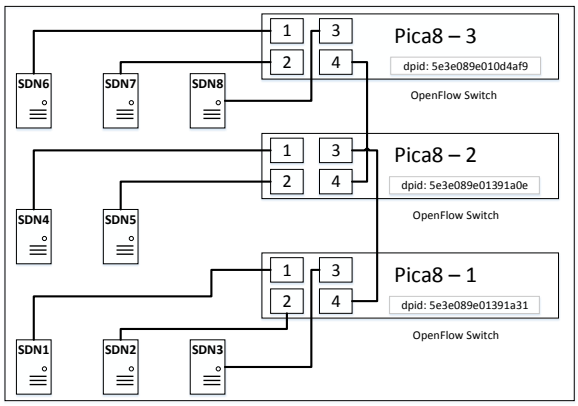

Fig. 7. Test Bed Data Plane Topology

two OpenFlow switches S_A and S_B, each of them maintains a single OpenFlow table, which are shown in TABLE I and TABLE II, respectively. A controller detects two elephant flows coming from port 21 in switch S_A and port 4 in switch S_B, respectively. These two flows demonstrate a certain level of similarity such that they are detected as a single FTE. If so, this elephant flow will match at least one FTE in both tables simultaneously.

For TABLE I, the potential FTE matching set is $F_A = \{2, i+1, i+2\}$ because all outputs of these FTEs consist of port 21 where elephant flow is observed. For TABLE II, the FTE set is $F_B = \{1, 3, j+1, j+2\}$. According to the definition of conjunction and disjunction, the match set of packets that matches at least one FTE in both F_A and F_B can be expressed as below:

$$\begin{aligned} M_{F_{A\wedge B}} =& M_{F_A} \wedge M_{F_B} \\ =& \left(M_2^A \vee M_{i+1}^A \vee M_{i+2}^A\right) \wedge \left(M_1^B \vee M_3^B \vee M_{j+1}^B \vee M_{j+2}^B\right) \end{aligned}$$

$M_{F_{A\wedge B}}$ consists of one or more match fields which determines the characteristics of matching packets, i.e., it is a HH match set.

IV. Heavy Hitter Identification

Once a suspected HH is found, the next step is to verify whether it is a real HH or a false positive. Our approach is to install a newly constructed FTE with the match fields as the HH match set into the switches where the HH occurs and then observe its statistics. One challenge is the behaviors of the existing FTEs cannot be affected by the new added flow entry. However, the newly added FTE usually shares the same match fields with the FTE with which the suspected HHs matches, and the HH will not match with the original FTE anymore if the new FTE is installed in the same table with higher priority. Thus, these FTEs must be carefully constructed to ensure the existing pipelines of all packets remain unchanged.

Using the "multi-table" feature provided by the latest OpenFlow specification [6], these new FTEs will be added in a separate table, table "0", to guarantee that all packets match against the FTEs in this table while preserving the original actions being executed. Table "0" is specifically designed for HH verification. The process is illustrated in Fig. 6 in which all the instructions of the FTEs in Table 0 have been set as "Goto Table: 1". In Table 0, all packets will be directed into next table for normal processing without performing any additional actions. The counters of these new FTEs in Table 0 are recorded in switch side. The controller retrieves these FTEs's statistics to validate whether a HH match set really represents an elephant flow.

V. Evaluation

A. Test bed & Traffic

A simple leaf-spine topology is adopted in our experiment which uses one Ryu OpenFlow 1.4.0 controller, three physical switches and eight hosts. Fig. 7 illustrates the topology of the data plane in which the spine switch ("Pica8-2") connects the other two leaf switches, "Pica8-1" and "Pica8-3", respectively. The spine switch forwards packets according to the destination MAC as match set, while leaf switches forward packets according to the destination IP address as match set. All hosts are in the same IP subnet and all FTEs are installed proactively in the switches to ensure connectivity between hosts.

To demonstrate the functionality of our proposed HH detection solution, we choose Distributed Internet Traffic Generator (D-ITG) [19] for traffic generation as it supports the following

customization: i) number of flows per physical port; ii) transmission rate of each flow; iii) variance of the transmission rate for each flow; and iv) protocol, duration, jitter, and delay of each flow. D-ITG produces packet level traffic with customizable packet interarrival time and packet size.

We implement a *distance-based HH detection system* (DHHDS) in Python with three independent threads for traffic statistics retrieval, data logging and HH detection. The traffic rate threshold T (see Eq.(1)) is adjusted to the mean traffic rate of all monitored flows. Likewise, the similarity threshold ST is also tuned to the average APCA distance. Thus, any two or more flows will be considered as a single HH if their traffic rates and APCA distance are below the average transmission rate and distance respectively.

The traffic data streams retrieved from all active physical ports are continuous in the time domain. In our evaluation, each stream is split into a series of discrete datasets with fixed time intervals. Recent research shows that elephant flows maintain their state for 20-40 minutes [20]. In this paper, the length of each dataset is set as 60 minutes to guarantee most of elephant flows be covered by two consecutive datasets.

B. Functional testing

A functional test checks the detection correctness by comparing the results for a given input traffic set against the desired outcome. In the functional test, we define seven sets of test traffic, each of them has various number of flows and HHs.

The process of detection in the functional test is explained with the help of Fig. 8. The controller retrieves the real-time outgoing traffic data of all physical ports and then analyze these data and their associated FTEs to determine whether there are some HHs. Suppose there is a time period in which one flow out of port 2 in switch A and another flow from port 1 in switch B exceed the average transmission rate simultaneously, which implies that both flows might contain HHs. If their APCA distance is also less than the average distance of all potential HHs, they will be assumed to be the same HH. This HH matches all the flow entries associated with these two ports. As illustrated in Fig. 8, the match set of output with port 2 in switch A and port 1 in switch B are "DMAC: 00:11:22:33:44:55" and "DIP: 192.137.1.1/32", respectively. Thus we can determine that this HH goes to a machine with MAC address as 00:11:22:33:44:55 and IP address as 192.137.1.1 by the conjunction operation of these two match sets. In the identification stage, a new FTE with the HH match set {*"DMAC: 00:11:22:33:44:55", "DIP: 192.137.1.1/32"*} as the match fields will be installed and monitored.

Table III shows the functional test results of DHHDS with seven different sets of input traffic. As shown in Table III, these flows follow certain packet size distributions and contain different percentages of HHs. The detection precision, also called positive predictive value (PPV), is determined by the true positive (TP) and false positive (FP):

$$PPV = TP/(TP + FP).$$

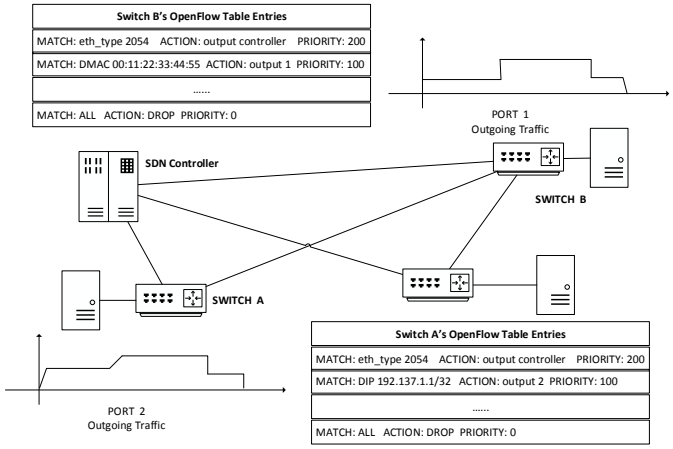

Fig. 8. One Heavy Hitter Detection Scenario

TABLE III
HEAVY HITTER FUNCTIONAL TEST RESULTS

Set	Num of flows	Num of HH	HH Traffic percentage	Packet size distribution	Precision
1	3	1	81%	Constant	100%
2	10	2	42%, 42%	Constant	50%
3	10	2	42%, 32%	Constant	75%
4	10	2	42%, 42%	Uniform $[8KB, 50KB]$	50%
5	10	2	42%, 32%	Uniform $[8KB, 50KB]$	75%
6	10	2	42%, 42%	Poisson $(61KB)$	50%
7	10	2	42%, 32%	Poisson $(61KB)$	75%

True positive means that a flow is correctly identified as HH while false positive incorrectly labels a flow as a HH. Our results show that the detection precision is dependent on each flow's transmission rate. If two different flows demonstrate a certain similarity in a time period, DHHDS might consider them as the same flow incorrectly. This explains the 50% detection precision in some cases because DHHDS cannot distinguish any two HHs with the same transmission rate. The scenario whereby two flows share the exact transmission rate is rare. Further improvements to the detection precision can be achieved by tuning the similarity threshold.

Another factor related to precision is the misidentification of an aggregate elephant flow as a HH. It happens when multiple flows share a common forwarding path between two intermediate switches, which can be avoided if we require at least one port in edge switches to be involved in the similarity computation. Another observation is that DHHDS is not sensitive to the traffic pattern. This is attributed to the APCA representation, which averages time series traffic and thus the distance between two APCA datasets are comparably insensitive to short traffic spikes.

The tests demonstrate that the HH detection using traffic

statistics and FTE functions is as expected. However, its efficiency needs to be evaluated to determine whether it can be deployed in a large network.

C. Efficiency Evaluation

In the functional test of DHHDS, a real testbed with physical switches and servers is used. However, the bottleneck in DHHDS is the communication overhead and computation capability of the controller. This is because the controller collects and analyzes statistics of all physical ports, which might affect the network performance as the network scales up. In this section, we first analyze the time overheads in a network with fixed number of ports and then repeat the evaluation with increasing the number of ports to understand the scaling behavior of DHHDS.

1) Evaluation Scenario: Since both factors are highly dependent on the traffic statistics which is measured on the level of physical ports, the number of active ports will be our evaluation units. Scaling to various number of switches and ports in our laboratory-level test bed is prohibitively expensive, therefore we simulate this scenario by replaying scaled up traffic statistics from our testbed (recall that DHHDS relies on traffic statistics from switches). In this performance evaluation, the controller is a desktop machine (Dell Optiplex 9020) with Intel Core i7-4790 CPU at 3.60GHz and 8Gb RAM running the Ryu controller. Besides, we also assume all switches' port are operating at 10Gbps and 10% of these ports contain heavy hitters.

2) HH Detection time: The time to complete a round of heavy hitter detection in DHHDS can be expressed as:

$$T_{HH} = T_{comm} + T_{switch} + T_{controller},$$

where T_{comm} represents the communication time between controller and switches, T_{switch} denotes the querying time for traffic statistics and $T_{controller}$ denotes the time to process traffic statistics at the controller side. Recent measurement studies have shown that both T_{comm} and T_{switch} are in order of milliseconds [21], [22] compared to seconds for $T_{controller}$, and can be safely ignored in computing measures of efficiency.

The processing time at the controller side, $T_{controller}$, is proportional to the number of active ports. In the scenario of a network with 4800–ports (100 switches$\times$48 ports), the computation time is 15.2 seconds. As shown in Fig. 9, when the number of active ports are increased from 10^2 to 10^5, the HH detection time increases linearly from less than one second to around five minutes. The five minute benchmark is the average time for HH detection for sampling-based methods [23] and this is shown as a horizontal line in Fig. 9.

Upon profiling the operations and computations in DHHDS, we found the most time-consuming procedures is APCA which costs more than 98% of the total computation time. This is also consistent with the fact that the complexity of APCA is $O(n\log(n))$ while the remaining procedures (smoothing, thresholding and windowing) are $O(n)$, where n denotes the length of sampled traffic data.

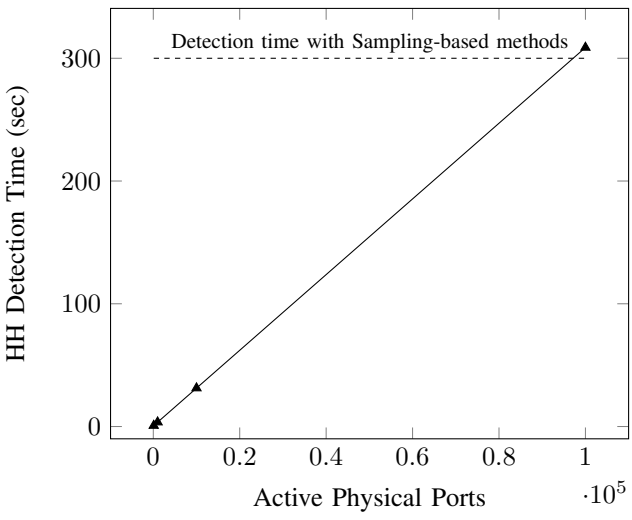

Fig. 9. HH detection time with increasing number of active ports.

3) Comparing different HH approaches: Table IV summarizes the major differences and indicative performances for three widely documented HH detection approaches in the 4800–port scenario. In this table, the values for computation time in the sampling based approach is calculated with the assumption that the link utilization rate is 20% and the sampling rate is 5% given by [24].

Comparing the communication overheads, it is clear that sampling-based solution is of limited use in large-scale networks. Sketch-based solutions have low communication overheads because they are self contained in individual switches, but it requires hardware customization making it less favorable with network operators. Our proposed solution leverages the traffic statistics rather than the packets and achieves a balance between the resource consumption and efficiency.

VI. Conclusion and Future work

Motivated by SDN, this paper tackles the problem of heavy hitter detection and identification without packet inspection. Our proposed solution relies on analysis of the readily available information in the controller and is especially suitable for large-scale SDN. Compared with traditional HH detection, our solution has three advantages: i) easily scalable; ii) arbitrary traffic detection without prior knowledge of traffic patterns; and iii) independent of underlying hardware design. The future work includes the deployment and evaluation of our solution on a realistic large-scale data center, which is now under negotiation with a local cloud service provider.

Appendix A
Representations

To reduce the dimensionality of the original data, a series of similarity search methods with a multi-dimensional index structure to index the data in the transformed space has been proposed [25]. Different representations determine the ease

TABLE IV
PERFORMANCE COMPARISONS OF HH DETECTION APPROACHES

	Objects Measured	Scale	Scanning(Computation) Time	Memory	Communication overhead
Sampling based	Packets	Multiple devices	∼510s (DFA OP [24])	7391KB (DFA OP [24])	Order of Gbps
Sketch based	Hash Table	Device-level	Line rate [9]	∼2MB [9]	Order of Kbps
Proposed DHHDS	Statistics	Multiple devices	∼15s	4915KB	Order of Mbps

and efficiency of similarity identification. Four indexable feature extraction techniques are simulated and verified with the windowed data: Discrete Fourier Transform (DFT), Discrete Wavelet Transform (DWT), Piecewise Linear Approximation (PLA) and Adaptive Piecewise Constant Approximation (APCA) [17].

An original signal is transformed into the frequency domain (DFT) or decomposed in terms of a basis set of functions (DWT). Each time series can be represented by a few selected coefficients corresponding to the low frequencies in DFT or scaling functions in DWT. These few coefficients preserve most of the energy of the original signal. The particular wavelet chosen in our comparison is the simplest wavelet form namely the Haar Wavelet. PLA projects a curve into a series of segmented straight-line. In our implementation, we use the least-square method (LSQ) to obtain an optimal line to fit the non-linear curve. Different from the aforementioned three representation techniques, APCA approximates each time series as a set of constant value segments of *varying* lengths.

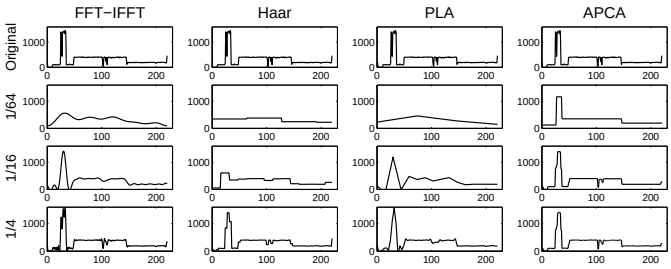

Fig. 10. The most common representations for time series data mining: DFT, DWT(Haar Wavelet), PLA, APCA

Taking the windowed dataset of sub-plot "Traffic Data Windowing-2" that appear earlier in Fig. 3 as an example, we illustrate the above mentioned four representations in Fig. 10. From left to right, the four columns demonstrate the signal transformation with algorithm DFT, DWT, PLA and APCA, respectively. The first row is the original signal, and the rest three rows are the representation with resampling rate at 1/64, 1/16 and 1/4, respectively. In our scenario, the size of the original dataset is 256, thus the number of resampling points from row two to row four in Fig. 10 are 4, 16 and 64, respectively. It can be observed that APCA outperforms the other three algorithms at the resampling rate 1/64 and 1/16 while all the feature selections except PLA reflect the main aspects of the original signal at rate 1/4. The varying length of each dimension in APCA fits well with elephant flows which usually last for a certain time in a relative stable volume, and lets APCA reflect the key features of traffic data with less storage space.

APPENDIX B
SET OPERATORS AND LOGIC OPERATIONS ON FTE

Let M_0 and M_1 denote a m-dimension match set $M_0 = \{f_0^0 \wedge f_1^0 \ldots \wedge f_m^0\}$ and n-dimension match set $M_1 = \{f_0^1 \wedge f_1^1 \ldots \wedge f_n^1\}$ respectively. The operations between M_0 and M_1 are defined as follows:

Definition 2 - Exact matching.

The physical meaning of two *exactly matching* FTEs is that one's matching flow will also match another one.

$$M_0 = M_1 : \forall i \exists j | [(f_i^0 = f_j^1) \wedge (f_i^1 = f_j^0)], f_{i,j}^0 \in M_0 \wedge f_{i,j}^1 \in M_1$$

Definition 3 - Subset (Superset).

The physical meaning of $M_0 \subset M_1$ is that M_0's matching flow does match M_1, but not vice versa. M_1 is also called a *superset* of M_0.

$$M_0 \subset M_1 : \forall i \exists j | [(f_i^0 \subset f_j^1) \vee (f_i^0 = f_j^1)], f_i^0 \in M_0 \wedge f_i^1 \in M_1$$

Definition 4 - Disjoint ($\overline{\bowtie}$).

The physical meaning of two *disjoint* FTEs is that one's matching flow will *NOT* match another and their respective matching packets share no common characteristics.

$$M_0 \overline{\bowtie} M_1 : \ \forall i, j \mid f_i^0 \wedge f_j^1 = \phi, \ f_i^0 \in M_0 \wedge f_i^1 \in M_1$$

Definition 5 - Joint ($\bowtie$).

The physical meaning of two *joint* FTEs is that their matching packets share one or more common characteristics, e.g., in the same IP subnet.

$$M_0 \bowtie M_1 : \ \exists i, j \mid f_i^0 \wedge f_j^1 \neq \phi, \ f_i^0 \in M_0 \wedge f_i^1 \in M_1$$

Three logical operations are used on match sets in the HH identification module (see Section IV): (i) Negation (not): $\neg$; (ii) Conjunction (and): $\wedge$; and (iii) Disjunction (or): $\vee$.
Negation is used in the set theory context and defined as follows:

$$\begin{aligned} \neg M_0 &= \neg\{f_0^0 \wedge f_1^0 \ldots \wedge f_m^0\} \\ &= \{\neg f_0^0 \vee \neg f_1^0 \ldots \vee \neg f_m^0\}. \end{aligned}$$

The conjunction operator on match sets M_0 and M_1 implies that packets of $M_0 \wedge M_1$ can match both sets simultaneously (as expressed in (5)).

$$M_0 \wedge M_1 = \begin{cases} M_0 \ or \ M_1 & \text{if } M_0 = M_1 \\ M_1 & \text{if } M_0 \subset M_1 \\ M_c\big(M_0^{'} \wedge M_1^{'}\big) & \text{if } M_0 \bowtie M_1 \\ M_0 \wedge M_1 & \text{if } M_0 \overline{\bowtie} M_1 \end{cases} \quad (5)$$

where M_c is the greatest common subset of M_0 and M_1, $M_0^{'}$ and $M_1^{'}$ are the subsets of M_0 and M_1, respectively, from which M_c has been removed.

The disjunction operator on match sets asserts that the matching packets of $M_0 \vee M_1$ match either M_0, M_1, or both as expressed in (6).

$$M_0 \vee M_1 = \begin{cases} M_0 \ or \ M_1 & \text{if } M_0 = M_1 \\ M_0 & \text{if } M_0 \subset M_1 \\ M_c\big(M_0^{'} \vee M_1^{'}\big) & \text{if } M_0 \bowtie M_1 \\ M_0 \vee M_1 & \text{if } M_0 \overline{\bowtie} M_1 \end{cases} \quad (6)$$

REFERENCES

[1] K.-C. Lan and J. Heidemann, "On the correlation of internet flow characteristics," Information Sciences Institute, University of Southern California, Tech. Rep. ISI-TR-574, July 2003.

[2] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker, "On the Characteristics and Origins of Internet Flow Rates," in *Proc of ACM SIGCOMM*, Pittsburgh, Pennsylvania, USA, 19-23 August 2002, pp. 309–322.

[3] R. Hofstede, P. Celeda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras, "Flow Monitoring Explained: From Packet Capture to Data Analysis with NetFlow and IPFIX," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 2037–2064, 20 Nov 2014.

[4] P. Phaal and M. Lavine, "sFlow Version 5," available online: http://www.sflow.org/sflow_version_5.txt, July 2004.

[5] C. Estan and G. Varghese, "New directions in traffic measurement and accounting," in *Proc of ACM SIGCOMM*, Pittsburgh, Pennsylvania, USA, 19-23 Aug 2002, pp. 323–336.

[6] "Openflow switch specification-version 1.4.0," 2013.

[7] Chakchai So-In, "A survey of network traffic monitoring and analysis tools," Washington University in St. Louis, Student project report, CSE576M Computer System Analysis (Fall 2006), 2009.

[8] A. Das, D. Nguyen, J. Zambreno, G. Memik, and A. Choudhary, "An fpga-based network intrusion detection architecture," *IEEE Trans on Information Forensics and Security*, vol. 3, no. 1, pp. 118–132, 2008.

[9] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with opensketch," in *Proc of the 10th USENIX Conference on Networked Systems Design and Implementation (NSDI)*, Lombard, IL, USA, 2-5 Apr 2013, pp. 29–42.

[10] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava, "Diamond in the rough: Finding hierarchical heavy hitters in multi-dimensional data," in *Proc of the ACM SIGMOD International conference on Management Of Data*, Paris, France, 13-18 June 2004.

[11] Y. Zhang, S. Singh, S. Sen, N. Duffield, and C. Lund, "Online identification of hierarchical heavy hitters: Algorithms, evaluation, and applications," in *Proc of the 4th ACM SIGCOMM Internet Measurement Conference (IMC)*, Taormina, Sicily, Italy, 25-27 Oct 2004, pp. 101–114.

[12] C. Estan, S. Savage, and G. Varghese, "Automatically inferring patterns of resource consumption in network traffic," in *Proc of ACM SIGCOMM*, Karlsruhe, Germany, 25-29 Aug 2003, pp. 137–148.

[13] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *Proc of the 9th ACM SIGCOMM Internet Measurement Conference*, Chicago, IL, USA, 4-6 Nov 2009.

[14] W. Willinger, "The discovery of self-similar traffic," in *Performance Evaluation: Origins and Directions*. Springer, 2000, pp. 513–527.

[15] A. Vahdat, M. Al-Fares, N. Farrington, R. N. Mysore, G. Porter, and S. Radhakrishnan, "Scale-out networking in the data center," *IEEE Micro*, vol. 30, no. 4, pp. 29–41, 2010.

[16] S. Muthukrishnan, *Data streams: Algorithms and applications*. Now Publishers Inc., 2005.

[17] K. Chakrabarti, E. Keogh, S. Mehrotra, and M. Pazzani, "Locally adaptive dimensionality reduction for indexing large time series databases," *ACM Trans on Database Systems*, vol. 27, no. 2, pp. 188–228, Jun 2002.

[18] R. Bifulco and F. Schneider, "Openflow rules interactions: definition and detection," in *Proc of the IEEE Software Defined Networks for Future Networks & Services (SDN4FNS)*, Trento, Italy, 11-13 Nov 2013, 6 pp.

[19] A. Botta, A. Dainotti, and A. Pescapè, "A tool for the generation of realistic network workload for emerging networking scenarios," *Computer Networks*, vol. 56, no. 15, pp. 3531–3547, 2012.

[20] K. Papagiannaki, N. Taft, S. Bhattacharyya, P. Thiran, K. Salamatian, and C. Diot, "A pragmatic definition of elephants in internet backbone traffic," in *Proc of the 2nd ACM SIGCOMM Workshop on Internet Measurment*, Marseille, France, 6-8 Nov 2002, pp. 175–176.

[21] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in *Proc of the USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE)*, San Jose, CA, 24 Apr 2012, 6 pages.

[22] R. Edwards, Y. Ye, A. Kaul, and J. Yu, "Characterization of SDN switch response time in proactive mode," in *Proc of the Open Networking Summit (ONS)*, Santa Clara, CA, USA, 2-4 March 2014, poster.

[23] G. R. Cantieni, G. Iannaccone, C. Barakat, C. Diot, and P. Thiran, "Reformulating the monitor placement problem: Optimal network-wide sampling," in *Proc of the 2nd ACM Conference on Future Networking Technologies (CoNEXT)*, Lisboa, Portugal, 4-7 Dec 2006, pp. 5:1–5:12. [Online]. Available: http://doi.acm.org/10.1145/1368436.1368444

[24] F. Yu, Z. Chen, Y. Diao, T. V. Lakshman, and R. H. Katz, "Fast and memory-efficient regular expression matching for deep packet inspection," in *Proc of the ACM/IEEE Symposium on Architecture for Networking and Communications Systems (ANCS)*, San Jose, California, USA, 4-5 Dec 2006, pp. 93–102.

[25] R. Agrawal, C. Faloutsos, and A. N. Swami, "Efficient similarity search in sequence databases," in *Proc of the 4th International Conference on Foundations of Data Organization and Algorithms (FODO)*, Chicago, Illinois, USA, 13-15 October 1993, pp. 69–84.