Received March 16, 2020, accepted March 31, 2020, date of publication April 14, 2020, date of current version April 30, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2987977

Flow-Aware Elephant Flow Detection for Software-Defined Networks

MOSAB HAMDAN[1], BUSHRA MOHAMMED[1], USMAN HUMAYUN[1], AHMED ABDELAZIZ[2], SULEMAN KHAN[3], M. AKHTAR ALI[3], MUHAMMAD IMRAN[4], (Member, IEEE), AND M. N. MARSONO[1], (Member, IEEE)

[1]Faculty of Engineering, School of Electrical Engineering, Universiti Teknologi Malaysia, Johor Bahru 81310, Malaysia
[2]Faculty of Computer Science, Future University, Khartoum 11115, Sudan
[3]Department of Computer and Information Sciences, Northumbria University, Newcastle upon Tyne NE1 8ST, U.K.
[4]College of Applied Computer Science, King Saud University, Riyadh 11451, Saudi Arabia

Corresponding authors: Mosab Hamdan (engmosab.hamdan@gmail.com) and M. N. Marsono (mnadzir@utm.my)

ABSTRACT Software-defined networking (SDN) separates the network control plane from the packet forwarding plane, which provides comprehensive network-state visibility for better network management and resilience. Traffic classification, particularly for elephant flow detection, can lead to improved flow control and resource provisioning in SDN networks. Existing elephant flow detection techniques use pre-set thresholds that cannot scale with the changes in the traffic concept and distribution. This paper proposes a flow-aware elephant flow detection applied to SDN. The proposed technique employs two classifiers, each respectively on SDN switches and controller, to achieve accurate elephant flow detection efficiently. Moreover, this technique allows sharing the elephant flow classification tasks between the controller and switches. Hence, most mice flows can be filtered in the switches, thus avoiding the need to send large numbers of classification requests and signaling messages to the controller. Experimental findings reveal that the proposed technique outperforms contemporary methods in terms of the running time, accuracy, F-measure, and recall.

INDEX TERMS Software-defined networking, flow classification, elephant flow detection.

I. INTRODUCTION

Software-defined networking (SDN) [1] has generated significant interest in industry and academia in recent years. The most crucial advantage of SDN is the opportunity to provide intelligence in computer networks. SDN capabilities include dynamic updates of the forwarding rules, software-based traffic analysis, and a logically centralized control network with a global view. These features allow the possible adoption of machine learning in network management [2]. However, the continuous growth of data traffic in terms of volume, velocity, and variety has made network traffic engineering a challenging task [3]. An accurate flow detection is vital for establishing appropriate forwarding strategies for various flow types, particularly for elephant flows (EFs) in an SDN environment.

Recent measurements conducted in data center networks [4], [5] have shown that 80% of the total flows take less than a few milliseconds and are less than 10 KB in size (i.e., mice flows, MFs), and that the majority of the traffic volume is accounted for by the top 10% of large flows (i.e., EFs). Any traffic that exceeds a certain threshold per unit time (e.g., 1 MBps) is often considered also an EF [6]. Given the high rate of EFs in network traffic, their effective control and rerouting can potentially improve the SDN network throughput [7]. By contrast, the competition for resources between MFs and EFs makes MFs to receive insufficient bandwidth [8]. Hence, EF detection [9] is an essential aspect of network traffic classification. The SDN controller does not need to process all flows, as the controller only needs to consider those EFs that severely impact the network performance when performing traffic management. If they are not efficiently managed, the network buffers can be filled with EFs, thereby leading to queuing delays and packet drops. Thus, EF detection is essential to easing network congestion [10].

Several EF detection techniques [6], [7], [11]–[20] have been previously proposed. However, these techniques are

The associate editor coordinating the review of this manuscript and approving it for publication was Haris Pervaiz.

preconfigured with fixed flow size thresholds in the switch, which can result in high rates of false positive and false negatives. Moreover, some methods require periodic extraction of the flow statistics (e.g., [6], [7], [16], [17]) or sampling packets (e.g., [11]–[15]) from SDN switches, result in a long flow detection latency and heavy controller-switch signalling overhead. Some other techniques require either important modifications in the switch hardware (e.g., [18]) or applying end-host inference (e.g., [19], [20]), which make adoption in existing SDN difficult to achieve.

By considering limitations as mentioned earlier, several improved EF detection techniques have been proposed [21]–[24]. However, these techniques are weakened by a slow convergence for several reasons, including the switch-controller interaction which requires a high bandwidth and long detection time. The thresholds of existing detection approaches are usually preconfigured without any consideration of the changing traffic load or distribution in the SDN networks, which may cause a high false detection. Flow detection in SDN also requires accurate real-time detection. Flow detection techniques based on statistical thresholds can operate in real-time but with a lower accuracy, and at the same time increasing the controller workload. This problem requires a careful trade-off balancing. When performing an SDN flow prediction, a failure to detect an EF can have more severe consequences than that for misdetection an MF. To further improve the accuracy of EF detection, the flow characteristics must be fully considered.

This paper presents a flow-aware EF detection technique for SDN. The proposed technique employs a pair of classifiers that run in tandem on the SDN switches and the controller, respectively, to share the tasks of classifying the EFs. Hence, most MFs can be filtered in the switches, and a large number of classification requests and signaling messages can be avoided at the controller. Our solution provides a good trade-off between the overall accuracy and the controller loads, which is critical for real-time traffic flow management. Several experiments have also been conducted on real datasets to measure the improvement in the controller running time, accuracy, F-measure, and recall. The key contributions of this paper are as follows.

- Proposing a flow-aware EF detection technique for SDN that can identify real-time EFs with low timing overhead and high detection accuracy, recall, and F-measure.
- Proposing a switch-side count-min (CM) sketch data stream structure used to filter MFs with commodity OpenvSwitch software. Moreover, the OpenFlow protocol is enhanced with extended signalling messages to handle the CM sketch data processing between switches and the controller side classifiers.
- Evaluating the performance of the classifiers in real-time for EF detection using real traces from the Internet and a data center in a Mininet simulation environment. The performance results show that our proposed technique can significantly improve the running time, accuracy, recall, and F-measure.

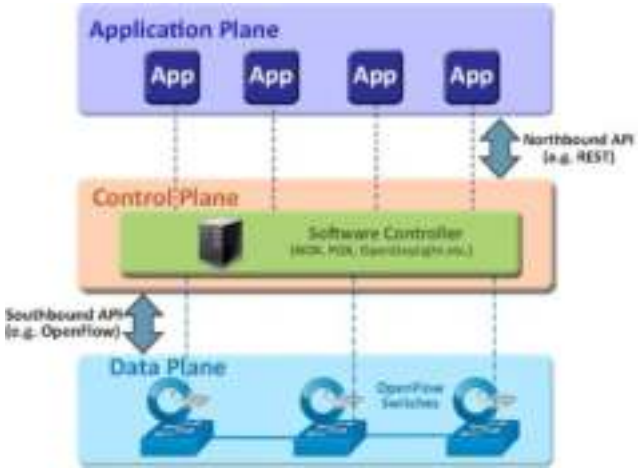


FIGURE 1. SDN architecture.

The reminder of this paper is organized as follows. A review of previous related studies is presented in Section II. Section III describes the framework design of the proposed EF detection technique. An evaluation of the results and a relevant discussion are detailed out in Section IV. Finally, Section V provides some concluding remarks regarding this research and areas of potential future study.

II. BACKGROUND AND RELATED WORK

A. SDN BACKGROUND

The SDN architecture characteristically abstracts the controller and data planes as separate entities, as illustrated in Figure 1. Programmability is the key characteristic of an SDN architecture allowing users to develop their customized applications. Using advanced policy applications and services and programmable application program interfaces (APIs) provided by the north-bound interface, users can develop applications of their choice at the application layer. In addition, the south-bound interface offers a standard API, such that the SDN controller communicates with two interfaces, including the south-bound and north-bound interfaces using the OpenFlow protocol [25], [26].

The SDN controller acts as a network operating system that views the network topology state comprehensively and manages OpenFlow switches through a secure communication channel [28]. Its responsibilities include managing and controlling how the switches process flows through the entries in the flow tables. Several variations of SDN (compatible) controllers have been developed, including the NOX controller [29], Ryu controller [30], and Floodlight controller [31]. Centralized control in SDN provides an architectural basis for open network programmability.

By providing a programmable interface for upper-level applications, the control plane can implement complex management functions such as EF detection strategies, load-balancing switches, and global monitoring of the network and its changing needs. OpenFlow switches forward messages based on the flow entry and various counters

defined for maintaining the traffic size or matching the number of data packets. These counters greatly simplify the collection of traffic statistics for EF traffic detection. The current factual south-bound protocol OpenFlow provides numerous control and monitoring mechanisms, which can flexibly implement flow management effectively and efficiently. With these features, EF detection in SDN has been rapidly advanced [9], [32].

B. OVERVIEW OF GENERIC FLOW CONTROL SCHEMES FOR DIFFERENT APPLICATIONS

A mechanism for controlling the flow of data between a pair of nodes is known as flow control. This is achieved by adjusting the transmission and receiving rates of the data. However, to bolster the quality-of-service (QoS) of the network with an improved quality-of-experience (QoE) for users, there is a need for an efficient traffic control strategy to cater for the ever-increasing traffic bandwidth [33].

The decision-making mechanism applied in the SDN architecture is the controller, which controls the functions for all flows within the entire SDN network [34]. Broadly, the flow control modes can be categorized into the coarse-grained [35], and fine-grained [36] controls. Several SDN-based flow control techniques have been proposed to further improve flow control. For example, in terms of traffic classification, Wang *et al.* [37] dealt with the traffic of unknown applications within the SDN by employing a semi-supervised machine learning approach in the classification of the QoS. The engine was able to run in real-time because only the first several packets of every flow were considered for feature extraction. Periodic polling for EF detection was also suggested in [7], which operates by extracting the per-flow statistics from its edge switches.

C. RELATED WORKS

The current EF detection techniques used in SDN fall into two main categories according to the detecting location: (i) switch-based detection and (ii) host-based detection.

1) SWITCH BASED DETECTION

A real-time EF detection system was proposed in [21]. The proposed method is comprised of two stages according to the statistical thresholding of the flow stream features. The first stage is to detect suspected EFs based on the statistical thresholds of multiple flows. The second stage is to identify EFs from the suspected EF set based on the features from the first few packets, which can offer timely and accurate flow classification. In addition, this approach employs a cost-sensitive learning approach using a C4.5 decision tree for real-time EF detection and the flow metric measurement. Chao *et al.* [23] presented an EF detection method based on a classification called FlowSeer. FlowSeer uses the features of the first five packets of a traffic flow to detect the EF. In FlowSeer, two classifiers are executed, one on the switch and the other on the controller. The switch-side classifier acts as a filter to remove most of the MFs, whereas the controller-side classifier verifies whether the EFs classified by the switch are genuine.

An EF prediction-mechanism was reported [24] for data center networks to address the characteristic traffic demands within the network. This approach seeks to reduce the overhead associated with the switch-to-controller communication by forecasting the EFs and adapting their routing policies in response to the ever-changing conditional demands on the network. However, this study has some scalability limitations when connecting to large-scale networks that are more complex and dynamic, such as multi-tenant cloud networks, large virtualized data centers, and Internet-of-Things (IoT). Huang *et al.* [22] proposed the arrangement of a pair of classifiers that respectively run on the controller and switch. Due to a limited switch computing power, only rules and decision trees classifiers can be used on the switch side. The controller, coupled with the switch-side classifier, accordingly rewrites the classifier rules and updates the switch flow table.

All approaches mentioned above can reduce the communication between the switches and the controller by keeping the frequency of transferring EFs statistics for flow setups to become minimal. Furthermore, by reducing the switch-controller communication, the workloads of both the controller and the network, which are the inherent overhead in the implementation of flow-based networking, can be reduced. In addition to fixing the threshold value, the EF detection on switches also requires modifying the switch hardware. Otherwise, the high detection accuracy of the EF detection system will be at the expense of a high network overhead, i.e., switch-controller to detect EF. Moreover, due to these limitations, achieving a balance among the accuracy, timeliness, and cost becomes difficult.

2) HOST-BASED DETECTION

Considering the scalability and timeliness of EF detection, the Mahout architecture [19] deploys a kernel patch in the terminal host to monitor the traffic statistics generated by the host and detect the EFs based on the pre-supposed EF threshold. To reduce the communication overhead, Mahout informs the controller regarding the EF and prescribes an in-band mechanism. Specifically, Mahout uses the differentiated service field of the IP header to mark the elephant stream. When the marked elephant stream reaches the switch, the switch forwards the corresponding packet towards the controller based on the default flow entry. Like Mahout, MicroTE [38] can conduct an analysis of all the network traffic.

By designating the monitoring end-host in each top-of-rack switch, the network traffic is collected, aggregated, and reported to the controller in time. However, due to the invisibility of network traffic generated by the virtual machines in the end-host, virtual traffic monitoring cannot be realized by simply deploying a kernel patch. Based on the monitoring tools such as VSFlow and NetFlow supported by OVS, EMC2 [20] recommends using a hypervisor deployed on the end hosts to collect the traffic statistics. However, the

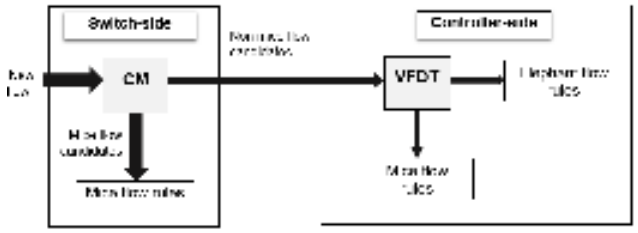


FIGURE 2. Proposed EF detection technique.

collected data needs to be sent to the centralized flow collector for further analysis, which may also result in an overhead of the monitoring traffic.

III. PROPOSED SWITCH-CONTROLLER FOR EF DETECTION

Elephant traffic detection in SDN must be fast, lightweight, and non-intrusive (i.e., its impact on the control plane should be minimal). At the same time, the detector must be able to accurately differentiate EFs from MFs for an effective flow migration based on a specific cost argument.

The EF detection process can be split between the controller and switches. Given the limited computational power of the switches, the classifier on the switch-side must be lightweight and designed such that it places more emphasis on a high recall. Therefore, the switch-side classifier can detect most of the EFs at the expense of false detecting some of the MFs. Meanwhile, the controller-side has more computational power and thus uses more features. Therefore, the classifier on the controller-side must place more emphasis on a higher F-measure and precision as opposed to the emphasis on recall for the classifier on the switch-side.

Figure 2 shows an operation of the proposed technique. When a new flow arrives, the switch-side classifier pre-filters the MFs based on the CM sketch algorithm [39]. The controller periodically trains the switch-side CM sketch model, emphasizing an optimal recall rate, which reduces the misdetection of potential (i.e., candidate) EFs. The candidate EFs are forwarded by the switch to the controller to performs the controller-side of the process. This is more of a practical streaming classification model using a very fast decision tree (VFDT) classifier. Once an EF is recognized, the CM training model is updated and converted into minimal sets of rules, given the limited nature of the flow table size of the switch. Figure 3 shows a flow chart of the EF detection.

The switch-side classifier is based on the CM sketch algorithm [39]. Because switches have limited computing power, the training of the classifier is achieved by either the controller or an off-line server. The CM sketch reports the state of its performance in terms of delay (i.e., buffer load), total number of packets handled, and list of hashed IP source-destination address pair for the EF candidates. A CM sketch algorithm used in the switch-side provides a quadruple of the hashed IP address, the number of packets, the aggregated packet sizes, and the average delay, which

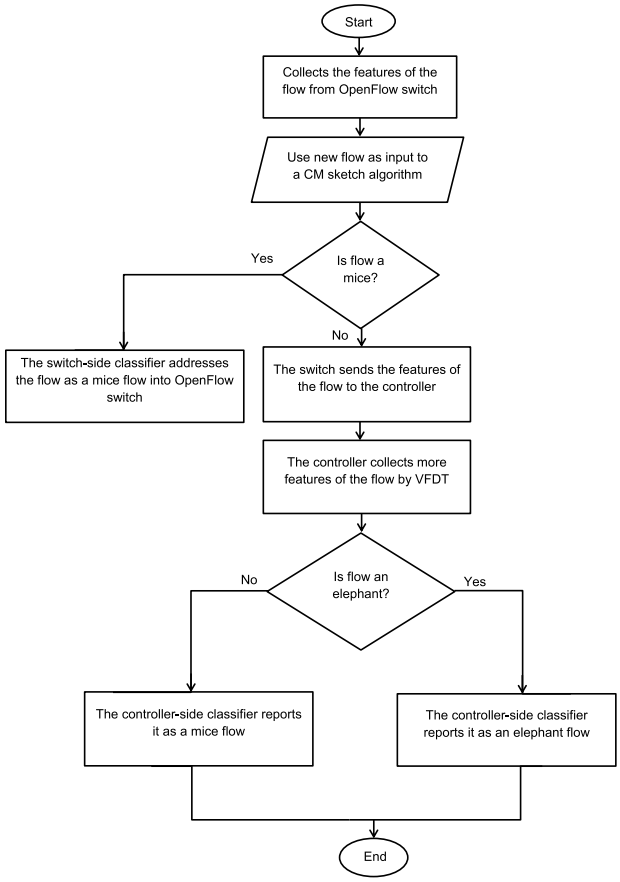


FIGURE 3. Flow chart of the EF detection technique.

is forwarded to the controller side. This approach also adds to the network traffic overhead as the switch-side classifier needs to communicate with the controller-side classifier.

The controller-side is created using a VFDT classifier based on the study in [40]. Because it is a multi-commodity flow problem, the tree needs to be trained using such data and based on the EF definition. For any given flow identified by an IP address pair, a set of alternative routes can be generated by any routing algorithm. Using the total packet arrival metrics from the switches, the controller predicts the network delay after a candidate EF passes. Based on these metrics, it selects the flow and path, which minimizes the functional cost (with parameters consisting of the maximum network delay and the number of hops). The selected flow and a new path are used to create migration instructions, which are compiled and sent to the switches.

A. SELECTION OF EF FEATURES

Before describing the EF detector architecture, we first need to define the EF and its features from the flow statistics.

1) DEFINITION OF THE EFs

Some studies, e.g., [7], [41] define an EF based on the bandwidth use over the specified limit for a specified time. The features used to accurately describe these properties are the

bandwidth and duration. These can be estimated based on the number of packets arriving per time window. Because the definition does not contain any information on the flow type, such features do not add any additional information regarding the flow. The time window in which the packets are captured includes the information on the average interarrival time of the packets.

2) SELECTION OF FEATURES FOR A FLOW CLASSIFICATION

It is highly desirable to use features that do not exhibit a correlation. Most of the models perform poorly in the presence of multicollinearity, i.e., when other variables can predict one explanatory variable [42], [43]. In addition, redundant variables increase the computational cost in terms of time and storage. The packet characteristics available are flow end-point identifiers i.e., the IP addresses and ports. The OpenFlow protocol indicating the type of flow is often associated with the port numbers. Therefore, the protocol type and port numbers are correlated. Similarly, properties related to Ethernet packets and IP packets are strongly correlated and therefore introduce multicollinearity.

In view of the load balancing flows in the data plane, the IP address pair is usually adequate to identify the path of the flow. Port numbers can be included to divide the flows into smaller sub-flows. In a large flow, one sub-flow is likely to be dominant. Because the whole flow may be subjected to rerouting, these sub-flows give little added information but increase the size and complexity of the implementation. An EF is also not characterized by its direction (as per the definition). Most flows are asymmetric and usually dominated by either uplink or downlink traffic (in terms of bandwidth and packets). The dominant link is, by definition, strongly correlated with the total flow. In the implementation, the IP source and destination addresses are hashed to form a flow identifier. The hash value is symmetric for the two IP addresses, and thus gives the same key regardless of direction. The distinction between the up-link and down-link packets double the number of flows and make the estimation computationally more expensive. Because only the dominant flows are of our interest, there is no reason to make this distinction.

Each additional feature increases the cost of the traffic classification (i.e., time and space) to both the switches and controller while maximizing the orthogonality when choosing the classification feature set. Thus, the feature set used for detection of the EF in an SDN environment has the following two attributes.

1) It is easy to extract using commodity OpenFlow switches. These features include the packet inter-arrival time, IP address and port number, packet size per flow, max and mean packet size, flow duration, and other flow statistics [44].
2) It is set up for a fast detection before a flow is concluded. For example, some features such as the frame length can only be obtained after the flow is concluded, thus failing to meet our needs for fast EF classification.

Therefore, we only consider the flow feature sets collected by inspecting the IP header. The flow contains the same five-tuple IP packets {protocol, src_port, dst_ip, src_ip, dst_port} with each flow distinguishable by statistical features such as the IP source and destination addresses, frame length, and average round trip time of a TCP Stream (TCP.analysis.ack_RTT) [44].

B. SWITCH-SIDE EF DETECTION

On the switch side, we use a CM sketch [39] to detect EF (heavy hitter) candidates. This method is fast and lightweight. As the CM sketch uses hashed IP address pairs, the IP address pair for the EFs also must be retrieved. The switch does not store or count the total number of flows, only the IP addresses. This sampling can be conducted at a relatively high frequency. After each sampling period, the result is stored in a data structure containing EFs and all packets. The arrival process vectors (containing the EF, and total flows), the flow identifiers (IP address pairs) of the EFs, and the buffer load data are sent to the controller. We next describe how the CM sketch technique works.

C. THE CM SKETCH

The CM or the Cormode-Muthukrishnan sketch is a data type suitable for counting frequencies, which is the frequency of arriving packets associated with a particular flow [39]. An end-to-end flow from a network perspective is defined by an IP address pair (source and destination IP addresses). A hash function of the IP address pair is used as an identifier in the sketch. For this purpose, it is convenient to use the IP range function in the Python package `iptools`. The hash is generated by the following:

```
h = iptools.IPRange (ipsrc, ipdst)
iphash = h.__hash__()
```

The first step in the CM sketch algorithm for finding EFs can now be formulated as finding the *heavy hitters* in the sketch, which are high-frequency flows among all available flows. The heavy hitter problem can be formulated as a sequence of point queries to the sketch, which returns an approximate frequency related to the index, which is the hashed pair of IP addresses. For this estimation, the approximation factor ϵ and failure probability δ are set at the sketch initialization, such that the estimate $\hat{\upsilon}_i$ of the true frequency υ_i of index i can be presented as Equation (1):

$$\hat{\upsilon}_i \leq \upsilon_i + \epsilon \|\upsilon\|_1 \tag{1}$$

with probability $1 - \delta$, and where $\|\upsilon\|_1 = \Sigma_{k=1}^{m} \upsilon_k$ is the L_1-norm. The L_1-norm is essentially unknown initially, and thus the approximation factor is treated as a fraction relative to the number of packets arriving in the switch. To initiate the sketch, the values of ϵ and δ determine the size, which is the width ω and depth d of the sketch as indicated in

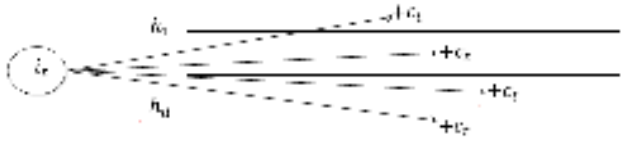


FIGURE 4. CM sketch data structure with $\omega = 9$ and $d = 4$ [39].

Equations (2) and (3), respectively:

$$\omega = \left\lceil \frac{e}{\epsilon} \right\rceil \tag{2}$$

$$d = \left\lceil \ln \delta^{-1} \right\rceil \tag{3}$$

The natural base e that can be chosen freely for all $e > 1$. The width and depth determine the size of the sketch, which is $\omega \times d$ words. This is illustrated in Figure 4, where each item i is mapped to one entry in each row j by the hash function h_j, and when an update of c_t to item i_t arrives, c_t is incremented for entry. The cost of such an update is only related to the depth d of the matrix.

The sketch uses a second hash function to reduce the required space of the sketch. Therefore, the index key is further hashed to fit into the sketch width ω. Given a prime number $p \geq \omega$, the hash function can be chosen using Equation (4):

$$h(x) = ((ax + b) \mod p) \mod \omega \tag{4}$$

where $a \in \{1, \ldots, p-1\}$ and $b \in \{0, \ldots, p\}$ are known as the c-universal family of hash functions [45].

Since collisions are unavoidable when using a small space to represent a much larger range of values (i.e., the hash of the IP addresses), it follows that $\hat{\upsilon}_i \geq \upsilon_i$ for all i. The depth d of the sketch is made up of d instances using different hash functions, and taking the minimum over d instances to give the value $\hat{\upsilon}_i$ closest to υ_i. The hashed IP addresses cannot be retrieved because collisions occur. It is of interest to capture other features related to the flows identified by high-frequency arrivals, i.e., the packet length and round-trip time, representing the delay. For this purpose, two additional sketches using the same hash functions as the frequency counting sketch are initiated to aggregate the packet lengths and round-trip times, respectively.

As shown in Algorithm 1, by letting the sketches run for the chosen capture time interval of τ, the frequency is estimated by the frequency sketch. Because the IP addresses of the hash cannot be reconstructed, the heavy hitters are found by point queries to the sketch by taking the hash of the source and destination IP addresses. When the estimated frequency exceeds a set fraction ϕ of the L_1-norm $\| \upsilon \|_1$, the IP address pairs are saved together with the minimum of d (the frequency estimates corresponding to their hash value). For the additional features, the maximum values in the d arrays are used, rather than the minimum to ensures that the worst possible characteristics are captured.

Algorithm 1 Extended CM Sketch

Given : Parameters ϵ, δ, τ and ϕ and a packet capture stream P.
Let : $I = \emptyset$ be the set of unique IP address pairs.
Initialize: Initiate the three sketches S_f, S_t and S_d for frequency, throughput and delay respectively, with $\omega = \left\lceil \frac{e}{\epsilon} \right\rceil$, $d = \lceil \ln \delta^{-1} \rceil$, and determine the prime p.
Output : The k heavy hitters represented by the source IP, destination IP, frequency (number of packets), throughput (sum of the packet lengths), and average RTT ACK (delay).

```
Generate hash functions (a, b, ω, p) according to Eq. (4) and set time t = 0;
while t < τ do
    for each incoming packet in P do
        Save IP address in I indexed by its hash function value;
        Update S_f, S_t, and S_d using the same hash defined initially;
    end
    Update t with a timestamp;
end
if t ≥ τ then
    for all IP address pairs in I, query S_f with the corresponding hash function (a, b, ω, p)
end
else if υ̂_i > φ ||υ||_1 then
    Save the IP address pair and υ̂_i into the same hash;
    Query S_t and S_d;
    Reset t= 0;
end
```

The output from the sketch is a set of relatively high-frequency flows (a large number of packets per time unit). However, an EF is typically defined as a flow with a large throughput for a specific duration of time. The sketch records the estimated performance and delay, the latter is a likely effect of such a flow, but does not consider these parameters when filtering out the flows. By using these sketches on short time intervals, aggregation makes it possible to estimate the frequency (i.e., the time aspects of the flow). After each time interval, the extracted candidate EF data are sent to the controller-side, and the sketches are reset for the next aggregation interval.

D. CONTROLLER-SIDE EF DETECTION

The VFDT is a stream-based data mining classification algorithm that incrementally builds it model as a tree by the division of nodes into a pair of streams of incoming data. The tree expands incrementally as more data arrives. Therefore, the candidate EF data are fed into the VFDT for flow classification based on the aggregated attributes. The VFDT is a suitable method because the classification tree is binary.

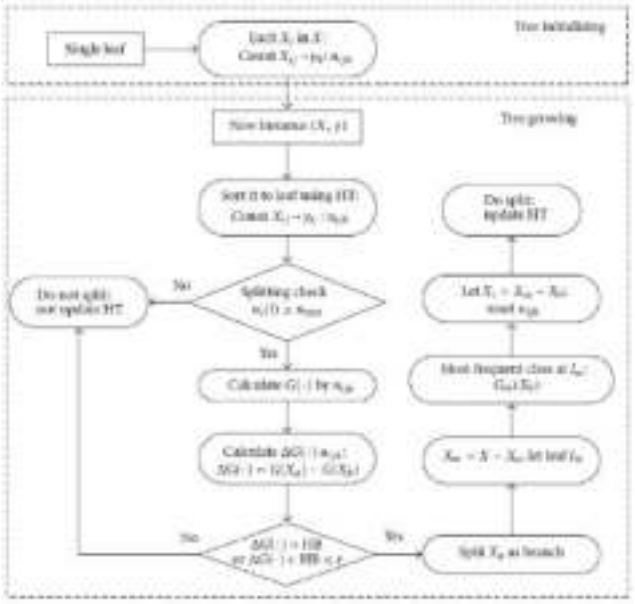


FIGURE 5. A flowchart representing the VFDT algorithm tree [46].

As with a sketch, the VFDT reads each candidate EF data point x only once and does not require the input data to be stored [40].

In the decision tree, each node represents an implementation of a logical test on a feature from the features of x. At the same time, each leaf indicates a classification from which an appropriate label y is assigned to the incoming data point x as $y = \text{VFDT}(x)$. The learning process of the tree is achieved through a successive replacement of each leaf with a node, starting from the root. The Hoeffding tree (HT) [40] algorithm uses the Hoeffding bound (HB) to train the model using the smallest possible number of training samples. The VFDT is made of key elements that include i) an initialization process of a tree beginning with a single leaf, and ii) a growth process of a tree where a repeated splitting check is heuristically carried out using the HB and an evaluation function $G(.)$. Information gain is used in VFDT to represent $G(.)$. Figure 5 shows the flow of operations in the VFDT algorithm [46].

The HB is the basis of the VFDT, where for a given sequence of independent random variables $0 \leq Xi \leq R$ bounded within the range R, the probability of the sample mean $\bar{X} = \frac{1}{n}(X_1 + X_2 + \cdots + X_n)$ deviating from its expectation $E(\bar{X})$ by a positive constant ϵ is related to the sample size n as indicated in Equation (5):

$$P(\bar{X} - E(\bar{X}) \geq \epsilon) \leq \exp(-2n\epsilon^2) \tag{5}$$

The HB states that, by consideration of n independent observations of a random variable with sample mean $\bar{r}$ and δ pre-defined tolerable estimation error, with probability $1-\delta$, the true mean of the variable is at least $\bar{r} - \epsilon$, where

$$\epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}} \tag{6}$$

The VFDT uses the HB to select the attribute to split as a decision node. Let x_a and x_b be the attributes with the highest and second-highest $G(.)$ respectively and $\Delta\overline{G} = \overline{G}(x_a) - \overline{G}(x_b) \geqslant 0$ be their difference. If $\Delta G > \varepsilon$ with $\mathbb{N}$ as the number of observed samples in the leaf, and $1 - \delta$ as the probability of x_a being the highest value attribute in $G(.)$ given by HB the leaf is then converted into a decision node splits on x_a. The HB is not dependent on the distribution of X_i, which is extremely convenient because the distributions of the traffic features are complex and vary with the application.

The HT algorithm aims to guarantee that the selected attribute with high probability and n examples is the same as that selected when using a significantly large number of examples. One major characteristic of the HT algorithm lies in the possibility of guaranteeing the construction of a tree that is asymptotically and arbitrarily comparable to the product of a batch learner. During each step, the attribute with the highest information gain is chosen as the test attribute. As the error ϵ decreases with increasing n, the difference in gain of the two attributes with the highest information decreases. When this difference falls below ϵ, the node is split, and testing on the attribute with the next highest information yields new leaves [40].

In the VFDT, the training sequence uses the EF definition based on the limits in throughput and duration, scaled to the time window used for aggregation as conducted by the CM-sketch. The maximum size of the tree is $2^{h+1}-1$, where h stands for the tree height, which equals the number of attributes. The VFDT thus produces a flow classification that can be used for processing and rerouting.

IV. EXPERIMENTAL RESULTS AND DISCUSSION

The experimental setup, results and in-depth discussion of our proposed methods are presented in this section. We also compare our findings with other contemporary methods found in the literature.

A. EXPERIMENT SETUP

The simulation was designed based on machine learning using Python socket programming APIs. The actual hardware used for the simulation included a Dell Inspiron laptop with a 3.20 GHz Intel i5-4570 CPU and 8 GB of RAM. In addition, Virtual Box was used as the virtual environment for loading a Mininet image. An SDN Hub 64-bit tutorial VM image is used to create a VM in Virtual Box with 4GB of RAM and a 20GB Hard drive. The Mininet image is a modified Ubuntu platform with a range of pre-installed and preconfigured network tools that include a Mininet simulation [47], and OpenvSwitch [48]. The Ryu controller [30] is installed and used as part of the SDN controllers for managing the OpenFlow compatible switches.

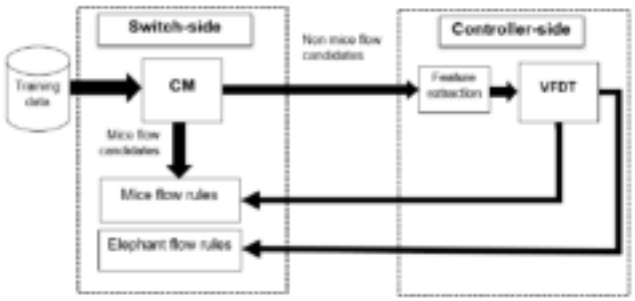


FIGURE 6. Experimental setup of the EF detection framework.

The three main steps applied in our experiment are shown in Figure 6. First, the data are divided into training and testing sets. Second, the training dataset is initially classified such that it can handle diverse varieties of traffic before given to the framework switches/controller sides to identify the correct attributes. Third, we train our CM sketch algorithm and VFDT classifiers to differentiate between EF and MF traffic, respectively. The performance of the proposed system is evaluated based on certain evaluation metrics.

1) DATASETS

We evaluate the proposed EF detection method on three different real network traffic datasets MAWI [49], UNI1 [49], and UNI2 [49]. The MAWI dataset (from April 9 to April 20, 2016) was obtained from a world-wide trace. The dataset comes from the daily tracking of trans-Pacific lines (the link was upgraded from 100 Mbps to 1 Gbps with a 150 Mbps committed access rate (CAR)). It has numerous stochastic factors, which makes the traffic classification more challenging. For this dataset, we select extensive flows as the significant flows because they dominate Internet traffic. This approach is used in our simulation for the measurement of the EF detection technique. A threshold of 10 MB/sec is set for the EF in this experiment [23]. In reality, the number of MFs is usually larger than the number of EFs. Thus, the MAWI dataset has approximately 10% - 20% EFs. The UNI1 and UNI2 datasets were captured from data centers studied in IMC 2010 [5]. EFs constitute ratios of approximately 2.5% and 5% in the UNI1 and UNI2 datasets, respectively.

2) EVALUATION METRICS

The performance metrics for the two-step flow classification method are the precision, accuracy, recall, F-measure, and running time. These metrics are all calculated from a confusion matrix.

In the confusion matrix, the true positive (TP) represents the number of actual positive records that are correctly classified. By contrast, the true negatives (TN) is the number of actual negative records correctly classified. In addition, the false positives (FP) is the number of misclassified negative records whereas the false negatives (FN) is the number of misclassified positive records.

1) The accuracy Acc is defined as the percentage of instances of the correct classification within the total number of instances.
$$Acc = \frac{TP + TN}{TP + FP + TN + FN} \tag{7}$$
2) The precision P is the total number of true positives divided by the sum of the false and true positives. The higher P reflects the lower number of false positives.
$$P = \frac{TP}{TP + FP} \tag{8}$$
3) The recall R is the number of true positives divided by the sum of the false negatives and true positives. A high R-value is desired.
$$R = \frac{TP}{TP + FN} \tag{9}$$
4) The F-measure is the harmonic mean of P and R, which has found widespread use in information retrieval and other supervised machine learning tasks. We also define high F values, as shown in Equation (10).
$$F = \frac{2P \times R}{P + R} \tag{10}$$
5) The running time T_{run} is the time taken to run a single experiment from start T_{start} to finish T_{finish}.
$$T_{run} = T_{start} - T_{finish} \tag{11}$$

B. EXPERIMENTAL RESULTS OF THE SWITCH-SIDE EF DETECTION

In this subsection, we compare a CM sketch method with C4.5 in terms of the accuracy, precision, recall, F-measure, and running time. We then discuss the trade-off of CM sketch.

1) CM PERFORMANCE COMPARISON

Table 1 compares the results of the CM sketch [39] and work proposed in [50] on the MAWI dataset. Hence, the results show that the CM sketch method outperforms C4.5 by eight times faster in terms of running time. Moreover, a values of $P = 1$ indicates zero false positives and an R of up to 90%. Our method performs better than the C4.5 method in terms of accuracy by up to 2.67%. The improved accuracy is because the estimated sum of the flow used by CM sketch is the hashed buckets with the smallest counter value. It can be determined whether a flow is a heavy hitter by checking whether its estimated sum falls below a certain threshold. The absolute change of flow over two epochs can be similarly used to verify whether a flow is a heavy hitter.

Moreover, our proposed switch-side CM sketch prefilter is real-time and has low-overhead. The primary rationale behind the proposed technique is to ensure the switches and the controller work together in sharing the EF classification task such that the majority of the MFs are filtered out by the CM sketch on switches, and the number of classification requests to the controller are significantly reduced. Table 2 demonstrates how our proposed flow detection method is different from other similar techniques.

TABLE 1. Performance evaluation of switch-side classifiers.

Dataset Distance measure	CM sketch [39]	C4.5 [50]
Average accuracy (%)	97.90	95.23
Running time (s/1000 flow instances)	0.054	0.43
Recall rate	0.90	0.59
Precision rate	1	0.39
F-measure value	0.95	0.47

TABLE 2. EF detection techniques in SDN.

Methods	Switch-side	Controller-side
EEFD [21]	—	C4.5
FlowSeer [23]	C4.5	CVFDT
EDMAR [22]	C4.5	C4.5
Proposed method	CM sketch	VFDT

2) SKETCH TRADE-OFF

CM sketch uses a hash function to count the frequency in a sub-linear space and store the number of occurrences in a stream into a $d \times w$ matrix. These parameters determine the trade-off between the accuracy and space/time constraints. Each row has an associated hash function. An arriving element is hashed, and its corresponding row is incremented by 1. Furthermore, the CM sketch solution might be lightly slower by waiting until the CM sketch has collected an adequate number of packets to form an aggregate to send to the controller side. This time overhead is noticeable if packets arrive at extremely irregular intervals.

C. EXPERIMENTAL RESULTS OF THE CONTROLLER-SIDE EF DETECTION

In this subsection, we present the results of a set of experiments conducted to validate the performance of our proposed method. First, we present the performance of the controller-side EF detection for the SDN network. We then compare it with other methods in terms of accuracy, precision, recall, F-measure, and running time.

1) CLASSIFICATION ACCURACY

Accuracy is one of the essential classifiers metrics. To evaluate the influence of our EF detection method, we tested its classification accuracy with several training data sizes ranging from 10,000 to 50,000 on the MAWI dataset. Figure 7 shows the accuracy of our purposed classification for various training sizes. We observed that our EF detection method on the controller-side achieves a higher accuracy than the existing EEFD method [21] by up to 0.7%, and the classification-based EDMAR [22] by up to 0.5%.

Figures 8(a) and 8(b) illustrate the precision and recall of our method compared to EDMAR [22], FlowSeer [23], and the Bayes network (BayesNet) [51]. Our method performs better in terms of accuracy, precision, and recall because the controller-side classifier becomes more accurate with an increase in the number of features used. Furthermore,

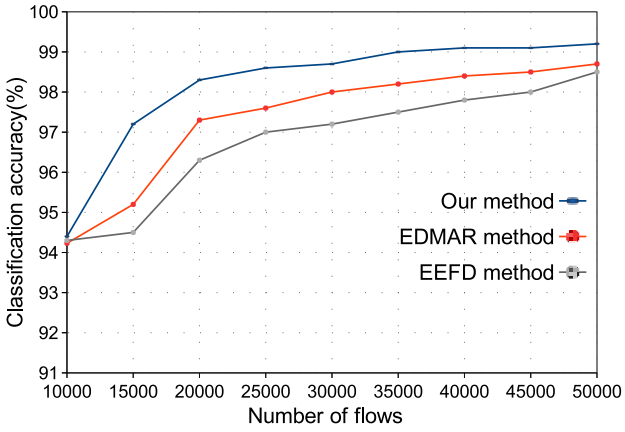
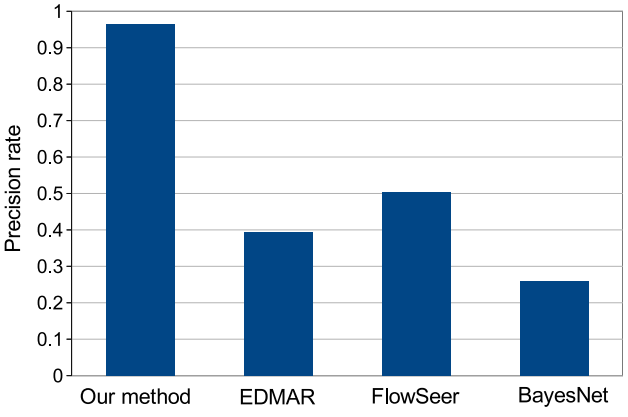
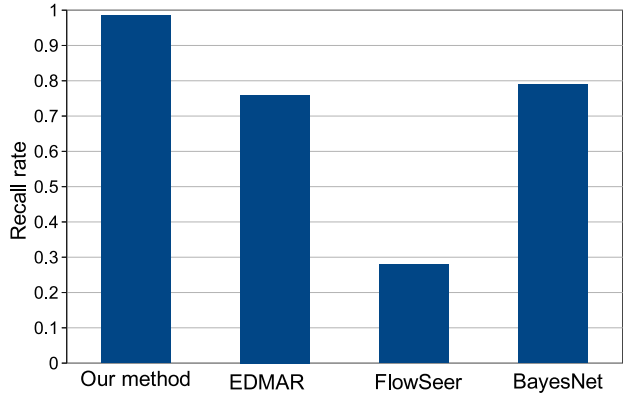


FIGURE 7. Accuracy with different numbers of flows.



(a) Precision



(b) Recall

FIGURE 8. (a) Precision rate and (b) recall rate comparison with different methods.

improvement in metrics is slightly due to the efficiency of the algorithm, early detection, and proper selection of features from accessible commodity switch features.

In our experiments, the EF detection applied to SDN achieves a recall rate of up to 98.3%. This high recall rate suggests that our method can detect most EFs, and only a few

TABLE 3. Overall performance comparison with existing EF detection methods using MAWI dataset, UNI1 dataset and UNI2 dataset.

Dataset Distance measure	MAWI Dataset				UNI1 Dataset		UNI2 Dataset	
	Proposed CM-VFDT	[22]	[23]	[21]	Proposed CM-VFDT	[23]	Proposed CM-VFDT	[23]
Average accuracy (%)	98.64	97.53	97.12	92.81	99.78	98.24	98.82	98.35
Running time (s/1000 flow instances)	0.07	0.11	0.23	0.62	0.06	0.43	0.09	0.52
Recall rate	0.98	0.77	0.28	_	0.99	0.98	0.98	0.75
Precision rate	0.98	0.35	0.56	_	0.98	0.97	0.97	0.57
F-measure value	0.98	0.48	0.37	_	0.98	0.97	0.97	0.65

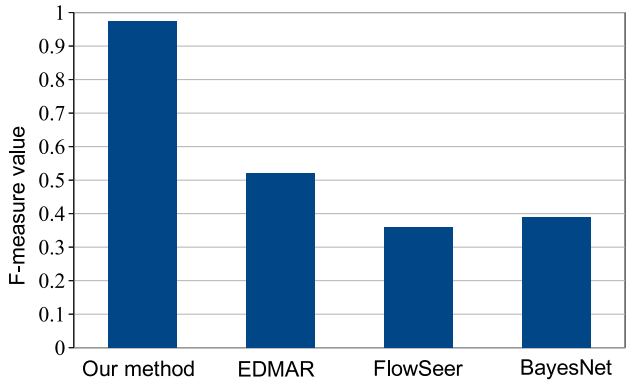


FIGURE 9. F-measure value comparison with different methods.

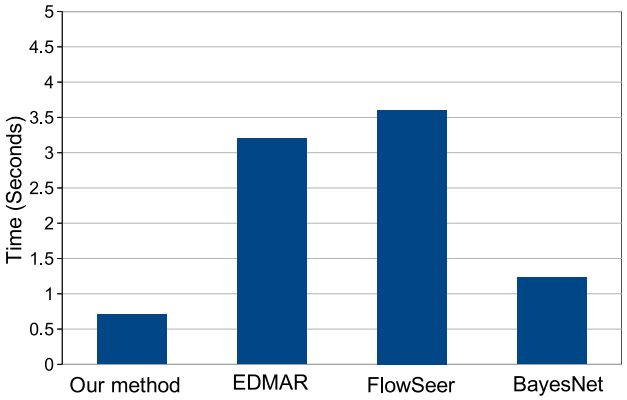


FIGURE 10. Running time comparison with different methods.

MFs are misidentified as EFs. A comparison of the F-measure between our method and other existing methods is shown in Figure 9. It can be observed that the F-measure of our method is over 96.1%, a significant improvement over the other methods.

2) VFDT PERFORMANCE COMPARISON

In terms of the amount of times required by different methods to detect EFs in SDN, our proposed method can detect EFs within the shortest time among the four methods (see Figure 10). Furthermore, our method takes less than a second to detect 10,000 flows, which is adequate to filter and detect EFs in the SDN network. The results show that the controller-side classifier function can achieve a better running time as it is lightweight as it does not store any dataset points in memory, making it ideal for the detection of EFs on the controller side. The decision tree model can be slowly built from scratch, which helps to detect EF at any point. Whenever a new data section arrives, the testing and training phase is carried out to keep the data stored up-to-date. It does not need to read the entire dataset and instead updates the decision tree to the latest incoming and collected statistical attributes, thereby consuming less memory. Furthermore, the use of the switch-side CM sketch classifier greatly minimizes flows as MFs by about 80%, while non-MF candidates can only give it to the controller-side by about 20%, further decreasing the controller-side load. These features make the VFDT a suitable candidate to introduce an autonomous decision-maker for the detection of EFs in SDN networks.

D. CM-VFDT PERFORMANCE COMPARISON

Table 3 shows the overall performance of our proposed method. We compared the performance of our EF detection method with that of the EDMAR method [22], FlowSeer [23], and the EEFD method [21]. The experiments were conducted on the MAWI dataset [52], UNI1 Dataset [49], and UNI2 Dataset [49]. We found that our method performs better than the other methods in recall and precision. Moreover, our method also performs better than other existing methods in terms of F-measure, indicating that our approach achieves a better balance between precision and recall. The results also show that this study achieves a higher accuracy and better running time.

The EF detection classifier was trained using a training flow, as described in Table 4, which summarizes the experimental results. The table shows the ability and efficiency of our method to detect EFs at the flow-level with an extremely small FN and high TP for all tests conducted on the switch/controller side. As the reason for these results, the CM sketch and VFDT methods apply traffic classification differently. By definition, EFs are specified on flows based on their duration and intensity (bandwidth), and the packet data have no information regarding the flow duration. Thus,

TABLE 4. Confusion matrix of flow-level of the MAWI dataset.

Dataset (1000 flows)	CM [39]		Proposed CM-VFDT	
EFs	TP	200	TP	194
	FN	21	FN	3
MFs	TN	779	TN	24
	FP	0	FP	0

the CM sketch creates an aggregate of packets to approximate the flow.

However, neither CM sketch nor VFDT has any long-term memory. Hence, to identify the EF, it is necessary estimate the duration by sampling the CM sketch and use it as an input to the VFDT classifier. Moreover, the improved metrics are due to the efficiency of the lightweight algorithm used on the switch-side to filter out most of the flows unlikely to be EFs. The experiments showed that the CM sketch algorithm is efficient at estimating frequencies of candidate EF with a fast update and query times, and low space usage. Finally, the switch-side only forwards the remaining potential EFs to the controller. The use of combined CM-VFDT greatly minimizes the classifier-side load. This technique proves to be a suitable candidate to introduce an autonomous decision-maker for the detection of EFs in SDN networks.

V. CONCLUSION

This paper presented a framework that can contribute to real-time traffic flow management in SDN networks. The proposed EF detection combines the switch-side extended CM sketch and controller-side the VFDT classifiers to provide real-time EF detection for effective and efficient SDN traffic flow management. The CM-VFDT classifiers can concurrently perform EF detection on commodity OpenFlow enabled switch and SDN controller. To detect EF candidates as heavy hitters, the fast and lightweight CM sketch classifier is used on the switches. The sketch extension focused on four flow features: delay (round-trip time), IP addresses, throughput, and packet count. The experiments are implemented in Mininet simulation using OpenvSwitch as OF switch managed by Ryu controller. Real traffic datasets such as MAWI, UNI1, and UNI2 are used to measure improvement in controller running time, accuracy, F-measure, and recall. Our experimental results show that EF detection method can achieve up to 98.13% accuracy with a higher recall rate and F-measure, and with better running time, which is better than other works in the comparative study.

Our directions for future works include a performance evaluation of the proposed algorithm in a broader orchestration context. With expansion growth of the SDN deployment in the near future, on-line flow classification will become more significant. Other applications that can benefit from flow classification include intrusion detection, load balancing, and bandwidth brokerage, for which the framework may need adaptation to accommodate data from various sensors.

REFERENCES

[1] "Software-defined networking: The new norm for networks," Open Netw. Found., ONF White Paper, 2012, vol. 2, pp. 2–6. Accessed: Mar. 12, 2020. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf

[2] J. Xie, F. R. Yu, T. Huang, R. Xie, J. Liu, C. Wang, and Y. Liu, "A survey of machine learning techniques applied to software defined networking (SDN): Research issues and challenges," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 1, pp. 393–430, 1st Quart., 2019.

[3] S. Mallesh, "Automatic detection of elephant flows through openflow-based openvswitch," M.S. thesis, Nat. College Ireland, Dublin, Ireland, Nov. 2017.

[4] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: Measurements & analysis," in *Proc. 9th ACM SIGCOMM Conf. Internet Meas. Conf. (IMC)*, 2009, pp. 202–208.

[5] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. 10th Annu. Conf. Internet Meas. (IMC)*, 2010, pp. 267–280.

[6] K. Lou, Y. Yang, and C. Wang, "An elephant flow detection method based on machine learning," in *Proc. 7th Int. Conf. Smart Comput. Commun.*, 2019, pp. 212–220.

[7] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. 7th USENIX Conf. Netw. Syst. Design Implement.*, 2010, pp. 89–92.

[8] W. Wang, Y. Sun, K. Salamatian, and Z. Li, "Adaptive path isolation for elephant and mice flows by exploiting path diversity in datacenters," *IEEE Trans. Netw. Service Manage.*, vol. 13, no. 1, pp. 5–18, Mar. 2016.

[9] B. Wang and J. Su, "A survey of elephant flow detection in SDN," in *Proc. 6th Int. Symp. Digit. Forensic Secur. (ISDFS)*, Mar. 2018, pp. 1–6.

[10] Y. Tian, J. Liu, Y.-X. Lai, Z.-S. Bao, and W.-B. Zhang, "TPEFD: An SDN-based efficient elephant flow detection method," *Chin. J. Netw. Inf. Secur.*, vol. 3, no. 5, pp. 70–76, 2017.

[11] C. Bi, X. Luo, T. Ye, and Y. Jin, "On precision and scalability of elephant flow detection in data center with SDN," in *Proc. IEEE Globecom Workshops (GC Wkshps)*, Dec. 2013, pp. 1227–1232.

[12] M. Afaq, S. Rehman, and W.-C. Song, "Large flows detection, marking, and mitigation based on sFlow standard in SDN," *J. Korea Multimedia Soc.*, vol. 18, no. 2, pp. 189–198, Feb. 2015.

[13] J. Suh, T. T. Kwon, C. Dixon, W. Felter, and J. Carter, "OpenSample: A low-latency, sampling-based measurement platform for commodity SDN," in *Proc. IEEE 34th Int. Conf. Distrib. Comput. Syst.*, Jun. 2014, pp. 228–237.

[14] Y. Afek, A. Bremler-Barr, S. Landau Feibish, and L. Schiff, "Sampling and large flow detection in SDN," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 5, pp. 345–346, Aug. 2015.

[15] F. Tang, H. Zhang, L. T. Yang, and L. Chen, "Elephant flow detection and differentiated scheduling with efficient sampling and classification," *IEEE Trans. Cloud Comput.*, early access, Feb. 26, 2019, doi: 10.1109/TCC.2019.2901669.

[16] C.-Y. Lin, C. Chen, J.-W. Chang, and Y. H. Chu, "Elephant flow detection in datacenters using OpenFlow-based hierarchical statistics pulling," in *Proc. IEEE Global Commun. Conf.*, Dec. 2014, pp. 2264–2269.

[17] W. Liu, W. Qu, Z. Liu, K. Li, and J. Gong, "Identifying elephant flows using a reversible MultiLayer hashed counting Bloom filter," in *Proc. IEEE 14th Int. Conf. High Perform. Comput. Commun. IEEE 9th Int. Conf. Embedded Softw. Syst.*, Jun. 2012, pp. 246–253.

[18] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling flow management for high-performance networks," *Comput. Commun. Rev.*, vol. 41, no. 4, pp. 254–265, Aug. 2011.

[19] A. R. Curtis, W. Kim, and P. Yalagandula, "Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection," in *Proc. IEEE INFOCOM*, Apr. 2011, pp. 1629–1637.

[20] V. Mann, A. Vishnoi, and S. Bidkar, "Living on the edge: Monitoring network flows at the edge in cloud data centers," in *Proc. 5th Int. Conf. Commun. Syst. Netw. (COMSNETS)*, Jan. 2013, pp. 1–9.

[21] P. Xiao, W. Qu, H. Qi, Y. Xu, and Z. Li, "An efficient elephant flow detection with cost-sensitive in SDN," in *Proc. 1st Int. Conf. Ind. Netw. Intell. Syst.*, 2015, pp. 24–28.

[22] Y.-H. Huang, W.-Y. Shih, and J.-L. Huang, "A classification-based elephant flow detection method using application round on SDN environments," in *Proc. 19th Asia–Pacific Netw. Operations Manage. Symp. (APNOMS)*, Sep. 2017, pp. 231–234.

[23] S.-C. Chao, K. C.-J. Lin, and M.-S. Chen, "Flow classification for software-defined data centers using stream mining," *IEEE Trans. Services Comput.*, vol. 12, no. 1, pp. 105–116, Jan. 2019.
[24] Z. Liu, D. Gao, Y. Liu, H. Zhang, and C. H. Foh, "An adaptive approach for elephant flow detection with the rapidly changing traffic in data center network," *Int. J. Netw. Manage.*, vol. 27, no. 6, Nov. 2017, Art. no. e1987.
[25] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "A roadmap for traffic engineering in SDN-OpenFlow networks," *Comput. Netw.*, vol. 71, pp. 1–30, Oct. 2014.
[26] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.
[27] M. P. Singh and A. Bhandari, "New-flow based DDoS attacks in SDN: Taxonomy, rationales, and research challenges," *Comput. Commun.*, vol. 154, pp. 509–527, Mar. 2020.
[28] S. Clayman, L. Mamatas, and A. Galis, "Efficient management solutions for software-defined infrastructures," in *Proc. NOMS - IEEE/IFIP Netw. Operations Manage. Symp.*, Apr. 2016, pp. 1291–1296.
[29] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in *Proc. 2nd USENIX Workshop Hot Topics Manage. Internet, Cloud, Enterprise Netw. Services*, 2012, pp. 1–6.
[30] *Ryu Controller*. Accessed: Jun. 1, 2018. [Online]. Available: http://osrg.github.com/ryu/
[31] *Floodlight Controller*. Accessed: Mar. 2 2018. [Online]. Available: http://floodlight.openflowhub.org/
[32] I. I. Awan, N. Shah, M. Imran, M. Shoaib, and N. Saeed, "An improved mechanism for flow rule installation in in-band SDN," *J. Syst. Archit.*, vol. 96, pp. 32–51, Mar. 2019.
[33] A. Jindal, G. S. Aujla, N. Kumar, R. Chaudhary, M. S. Obaidat, and I. You, "SeDaTiVe: SDN-enabled deep learning architecture for network traffic control in vehicular cyber-physical systems," *IEEE Netw.*, vol. 32, no. 6, pp. 66–73, Nov. 2018.
[34] P. Wang, H. Xu, L. Huang, C. Qian, S. Wang, and Y. Sun, "Minimizing controller response time through flow redirecting in SDNs," *IEEE/ACM Trans. Netw.*, vol. 26, no. 1, pp. 562–575, Feb. 2018.
[35] M. Xia, M. Shirazipour, Y. Zhang, H. Green, and A. Takacs, "Optical service chaining for network function virtualization," *IEEE Commun. Mag.*, vol. 53, no. 4, pp. 152–158, Apr. 2015.
[36] Z. A. Qazi, J. Lee, T. Jin, G. Bellala, M. Arndt, and G. Noubir, "Application-awareness in SDN," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 487–488, Sep. 2013.
[37] P. Wang, S.-C. Lin, and M. Luo, "A framework for QoS-aware traffic classification using semi-supervised machine learning in SDNs," in *Proc. IEEE Int. Conf. Services Comput. (SCC)*, Jun. 2016, pp. 760–765.
[38] T. Benson, A. Anand, A. Akella, and M. Zhang, "MicroTE: Fine grained traffic engineering for data centers," in *Proc. 7th Conf. Emerg. Netw. EXperiments Technol. (CoNEXT)*, 2011, pp. 1–8.
[39] G. Cormode and S. Muthukrishnan, "An improved data stream summary: The count-min sketch and its applications," *J. Algorithms*, vol. 55, no. 1, pp. 58–75, Apr. 2005.
[40] P. Domingos and G. Hulten, "Mining high-speed data streams," in *Proc. 6th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, 2000, pp. 71–80.
[41] K. Xi, Y. Liu, and H. J. Chao, "Enabling flow-based routing control in data center networks using probe and ECMP," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Apr. 2011, pp. 608–613.
[42] T. T. T. Nguyen and G. Armitage, "A survey of techniques for Internet traffic classification using machine learning," *IEEE Commun. Surveys Tuts.*, vol. 10, no. 4, pp. 56–76, 2008.
[43] B. Ghojogh, M. N. Samad, S. Asif Mashhadi, T. Kapoor, W. Ali, F. Karray, and M. Crowley, "Feature selection and feature extraction in pattern analysis: A literature review," 2019, *arXiv:1905.02845*. [Online]. Available: http://arxiv.org/abs/1905.02845
[44] *OpenFlow V1.4*. Accessed: Jan. 6, 2018. [Online]. Available: https://www.opennetworking.org/
[45] J. L. Carter and M. N. Wegman, "Universal classes of hash functions," *J. Comput. Syst. Sci.*, vol. 18, no. 2, pp. 143–154, Apr. 1979.
[46] H. Yang and S. Fong, "Moderated VFDT in stream mining using adaptive tie threshold and incremental pruning," in *Proc. 13th Int. Conf. Data Warehousing Knowl. Discovery*, 2011, pp. 471–483.
[47] *Mininet*. Accessed:Jun. 5, 2016. [Online]. Available: http://mininet.org/
[48] A. Al-Shabibi, M. De Leenheer, M. Gerola, A. Koshibe, G. Parulkar, E. Salvadori, and B. Snow, "OpenVirteX: Make your virtual SDNs programmable," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw. (HotSDN)*, 2014, pp. 25–30.
[49] *Data Center Measurement*. Accessed: Feb. 3, 2018. [Online]. Available: http://pages.cs.wisc.edu/~tbenson/IMC10_data.html/
[50] P. Domingos, "MetaCost: A general method for making classifiers cost-sensitive," in *Proc. 5th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, 1999, pp. 155–164.
[51] A. W. Moore and D. Zuev, "Internet traffic classification using Bayesian analysis techniques," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 33, no. 1, pp. 50–60, Jun. 2005.
[52] *MAWI Working Group Traffic Archive*. Accessed: Jun. 7, 2018. [Online]. Available: http://mawi.wide.ad.jp/mawi

MOSAB HAMDAN received the B.Sc. degree in computer and electronic systems engineering from the University of Science and Technology, Sudan, in 2010, and the M.Sc. degree in computer architecture and networking from the University of Khartoum, Sudan, in 2014. He is currently pursuing the Ph.D. degree with the Faculty of Engineering, School of Electrical Engineering, Universiti Teknologi Malaysia. His current research interests are in software defined networking (SDN), load balancing, network traffic classification, and future networks.

BUSHRA MOHAMMED received the B.Sc. and M.Sc. degrees in computer engineering and networks from the Faculty of Engineering and Technology, University of Gezira, Sudan. He is currently pursuing the Ph.D. degree with the School of Electrical Engineering, Universiti Teknologi Malaysia. He is currently a Lecturer with the Faculty of Computer and Statistics Studies, University of Kordofan. His research interests include computer architecture, network traffic classification and control, artificial intelligence, and optimization techniques.

USMAN HUMAYUN received the B.Sc. degree in computer engineering from the NFC Institute of Engineering and Technology, Multan, Pakistan, in 2008, and the M.Sc. degree in computer engineering from the Islamia University of Bahawalpur, Pakistan, in 2015. He is currently pursuing the Ph.D. degree with the School of Electrical Engineering, Universiti Teknologi Malaysia. He has been working as a Lecturer with the Department of Computer Engineering, Bahauddin Zakariya University, Multan, since 2009. His research interests are software defined networking (SDN), flow tables architectures, and controller overheads in SDN.

AHMED ABDELAZIZ received the M.Sc. degree in computer science and the Ph.D. degree in information technology from the Universiti Malaya (UM), Malaysia, in 2007 and 2017, respectively. He has been working on ONOS and OpenStack, since October 2015, during the Ph.D. degree research project. In the Ph.D. degree research, he proposed a novel service-based load balancing technique to use in the cloud using SDN and OpenStack. He is currently a full-time Assistant Professor with Future University (FU), Sudan. He published a number of ISI index articles in the areas of SDN, OpenFlow, and network virtualization. He has been involved in the Centre for Mobile Cloud Computing Research (C4MCCR) Projects funded by the Malaysian Ministry of Higher Education. His areas of interest include SDN/NFV technology, OpenStack, and network virtualization.

SULEMAN KHAN received the Ph.D. degree (Hons.) in computer science and information technology from the Universiti Malaya, Malaysia, in 2017. He was a Faculty Member of the School of Information Technology, Monash University, Malaysia, from June 2017 to March 2019. He is currently a Faculty Member of the Department of Computer and Information Sciences, Northumbria University, Newcastle upon Tyne, U.K. He has published more than 50 high-impact research articles in reputed international journals and conferences. His research areas include, but are not limited to, network forensics, software-defined networks, the Internet-of-Things, cloud computing, and vehicular communications.

M. AKHTAR ALI received the Ph.D. degree in computer science from Manchester University, in 2003. He is currently a Faculty Member of the Department of Computer and Information Sciences, Northumbria University, U.K. He served with this university for more than 19 years and has been involved in various research projects. His research interests include data analytics, databases, and machine learning.

MUHAMMAD IMRAN (Member, IEEE) received the Ph.D. degree in information technology from the Universiti Teknologi PETRONAS, Malaysia, in 2011. He is an Associate Professor with the College of Applied Computer Science, King Saud University, Saudi Arabia. His research was financially supported by several grants. He has completed a number of international collaborative research projects with reputable universities. He has published more than 200 research articles in peer-reviewed, well-recognized international conferences and journals. His many research articles have been highly cited and frequently downloaded. His research interests include mobile and wireless networks, the Internet of Things, big data analytics, cloud computing, and information security. He was consecutively awarded as an Outstanding Associate Editor of IEEE Access, in 2018 and 2019, in addition to numerous other awards. He has been involved approximately in 100 peer-reviewed international conferences and workshops in various capacities such as a chair, co-chair, and technical program committee member. He served/serving as a Guest Editor for approximately two dozens special issues in journals such as the *IEEE Communications Magazine*, the *IEEE Wireless Communications Magazine*, *Future Generation Computer Systems*, IEEE Access, and *Computer Networks*. He has served as an Editor-in-Chief for the European Alliance for Innovation (EAI) *Transactions on Pervasive Health and Technology*. He is currently serving as an Associate Editor for top-ranking international journals such as the *IEEE Communications Magazine*, the *IEEE Network*, *Future Generation Computer Systems*, and IEEE Access.

M. N. MARSONO (Member, IEEE) received the B.Eng. degree in computer engineering and the M.Eng. degree in electrical engineering from the Universiti Teknologi Malaysia, Malaysia, in 1999 and 2001, respectively, and the Ph.D. degree in electrical and computer engineering from the University of Victoria, Victoria, BC, Canada, in 2007. He is an Associate Professor in electronics and computer engineering with the Faculty of Engineering, School of Electrical Engineering, Universiti Teknologi Malaysia. His research interests are in many-core system-on-chips, network-on-chip interconnects, domain-specific computer architectures, network processing algorithmics, and network processing accelerators.