SPECIAL ISSUE PAPER

An adaptive approach for elephant flow detection with the rapidly changing traffic in data center network

Zehui Liu1 | Deyun Gao1 | Ying Liu1 | Hongke Zhang1 | Chuan Heng Foh2

1School of Electronic and Information Engineering, Beijing Jiaotong University, Beijing, China
2Department of Electrical and Electronic Engineering, University of Surrey, Surrey, UK

Correspondence
Ying Liu, School of Electronic and Information Engineering, Beijing Jiaotong University, Beijing 100044, China.
Email: yliu@bjtu.edu.cn

Funding information
973 Program, Grant/Award Number: 2013CB329100; Fundamental Research Funds for the Central Universities, Grant/Award Number: 2015JBM008

Summary
Software-defined network, which separates control plane from the underlying physical devices, has the advantages of global visibility and high flexibility. Among the most typical applications in software-defined network, there is significant interest on classifying flows, especially for elephant flow detection. Previous studies show that detecting and rerouting elephant flows (flows that transfer significant amount of data) effectively can lead to a 113% improvement in aggregate throughput compared with the traditional routing. However, the threshold of the existing detection approach was preconfigured without the consideration of the rapidly changing traffic in data center networks. This phenomenon could cause high detection error rate. To address this problem, we propose an adaptive approach for elephant flow detection, which could efficiently identify elephant flows with low latency and low overhead. Particularly, to meet the demands of the traffic characteristics in data center networks, dynamical traffic learning algorithm is adopted to configure the threshold value real timely and dynamically. Numerical results and experimental tests show that the mean error rate of detection is only 4.61% and the maximum number of packet-in messages is minimum compared to other methods.

1 | INTRODUCTION

Software-defined network (SDN) has emerged as an active field in both industry and academia. The most significant feature of SDN is to provide a global view of the network and separate control plane from data-forwarding plane. OpenFlow, as the most common way to achieve SDN, is an open standard, which allows the network innovation to be deployed through experiments by the researchers. In contrast to traditional networks, there is a centralized controller to enforce the management of an entire network, where operators have to modify functionality by complicated configuration. Furthermore, the centralized controller maintains all the switches states in the network, calculates routing paths, and sets up rules along the paths to forward packets. This kind of flexibility not only eliminates many issues in traditional networks but also provides a flow-level control of network traffic. As the above contributions, the centralized network control has also been considered in high-performance networks, such as data center networks.

However, there is a bottleneck that control plane is hard to scale up to operate the rapidly changing traffic in data center. According to previous measurements in data center, it is stated that 80% of the flows are smaller than 10 KB and last under a few milliseconds while top 10% elephant flows account for most of the traffic volume. Then, the arrival and departure of flows are very fast, which causes the difficult for controller to allocate resource for every new flow. In the meantime, a large number of
flows invoke controller too frequently because the switches always regard them as new flows. The high workload in controller

To reduce the workload and improve the efficiency on control plane, many elephant flow detection approaches had been pro-

Motivated by above analysis, we propose an adaptive approach for elephant flow detection by adopting dynamical traffic

The main contributions of this paper are as followed:

The rest of this paper is organized as follows. Section 2 makes an overview of the proposed architecture and related work.

2 | BACKGROUND AND RELATED WORK

2.1 | SDN and OpenFlow

Software-defined network is an emerging networking paradigm that gives hope to change the limitations of current network

As shown in Figure 1, we define an SDN as a network architecture with 4 pillars. First, the control and data planes are
decoupled. Control functionality is removed from network devices that will become simple packet-forwarding elements. Second,
forwarding decisions are flow based, instead of destination based. A flow is broadly defined by a set of packet field values acting
as a match criterion and a set of actions. In the SDN context, a flow is a sequence of packets between a source and a destination.
All packets of a flow receive identical service policies at the forwarding devices. The flow abstraction allows unifying the
behavior of different types of network devices, including routers, switches, firewalls, and middleboxes. Flow programming
enables unprecedented flexibility, limited only to the capabilities of the implemented flow tables. Third, control logic is moved
to an external entity, the so-called SDN controller or NOX. The NOX is a software platform that runs on commodity server
technology and provides the essential resources and abstractions to facilitate the programming of forwarding devices based on a logically centralized, abstract network view. Its purpose is therefore similar to that of a traditional operating system. Fourth, the network is programmable through software applications running on top of the NOX that interact with the underlying data plane devices. This is a fundamental characteristic of SDN, considered as its main value proposition.

A number of protocol standards exist on the use of SDN in real applications. One of the most popular protocol standards is called OpenFlow. OpenFlow is a protocol that enables the implementation of the SDN concept in both hardware and software. An important feature of OpenFlow is that scientists can use the existing hardware to design new protocols and analyze their performance. Now, it is becoming part of commercially available routers and switches as well. As a standard SDN protocol, OpenFlow was proposed by Stanford. Regarding testbeds of OpenFlow, many designs have been proposed for OpenFlow protocols. They use open source codes to control universal SDN controllers and switches. Regarding switches, OpenVSwitch (OVS) is one of the most popular, software-driven OpenFlow switch. OpenFlow is flow-oriented protocol and has switches and ports abstraction to control the flow. In OpenFlow, there is a software named controller that manages the collection of switches for traffic control. The controller communicates with the OpenFlow switch and manages the switch through the OpenFlow protocol. An OpenFlow switch can have multiple flow tables, a group table, and an OpenFlow channel, as shown in Figure 2. Each flow table contains flow entries and communicates with the controller, and the group table can configure the flow entries. OpenFlow switches connect to each other via the OpenFlow ports.
Initially, the data path of the OpenFlow routing devices has an empty routing table with some fields (such as source IP address and QoS type). This table contains several packet fields such as the destination of different ports, as well as an action field that contains the code for different actions, such as packet forwarding or reception. This table can be populated on the basis of the incoming data packets. When a new packet is received that has no matching entry in the data flow table, it is forwarded to the controller to be processed. The controller is responsible for packet handling decisions, for example, a packet is either dropped, or a new entry is added into the data flow table on how to deal with this and similar packets received in the future.

2.2 | Overhead

In this section, we mainly analyze the detection characteristics of the previous methods proposed by some researchers. By theoretical analysis the overhead and accuracy of these detection approaches, we show that our solution can scale well to support data centers while ensuring the accuracy of elephant detection and meeting the demands of traffic characteristics in data center networks.

1. Pull-based statistics: OpenFlow provides 3 kinds of statistic counters (packets, bytes, and duration time) for each flow. The central controller maintains these counters and network state by sending a Read-State message periodically to data-plane. Since detection needs to pull all the flow statistics in each edge switch to central controller, it could lead to high communication and processing overhead. Although, Lin et al. propose a hierarchical statistics pulling (HSP) mechanism to detect elephant flows in data centers by using a combination of aggregate and individual statistical message in the OpenFlow protocol. Furthermore, Lin et al. present an elephant-sensitive HSP by adding 2 supplement functions called elephant store and range splitting to improve the performance of HSP. However, they are not suitable for data center, due to their high cost and low accuracy with fixed threshold value.

2. Pulling statistics: It using OpenFlow needs to establish flow entry table for all flow, and each request returns 88 bytes to controller that will occupy the limited bandwidth between control plane and data plane. Considering a data center with 100 edge switches, it produces 10 million flows per second on average. If we apply pull-based method to collect statistics and identify elephant flows, it would return 6.5 GB monitoring messages under the condition that control plane has sufficient capacity to cope with this heavy traffic. Although HSP improves the performance of ISP, the method uses the fixed threshold value to detect elephant flow without considering the rapidly changing traffic in data center networks.

3. Switch-trigger: Instead of monitoring flows in control plane, this solution sets up a sniffer agent or applications at the switch. The proposed method shows that effective flow management can be achieved by devolving control of most flows back to the switches, while the controller maintains control over only targeted significant flows and has visibility over only these flows and packet samples. It reduces the switch-internal communication between control-planes and data-planes by reducing the need to transfer statistics for boring flows, and potentially reducing the need to invoke the control-planes for most flow setups. It therefore reduces both the intrinsic and implementation overheads of flow-based networking, by reducing load on the network, the switch control-plane, and the central controller. Switch-trigger method not only needs to modify hardware but also has the fixed threshold value. Otherwise, this elephant flow detection system has low detection rate and low network overhead.

4. Sampling: Sampling is a universal approach to obtain traffic measurement profiling. Packet sampling system, sFlow, has been supported by a range of device vendors. The sFlow method is scalable to the traffic volume, especially under the frequently changing traffic of data center. Comparing to capturing every flow statistics, the agent samples the packets 1 out of k packets and only sends the sampled header message to the central collector. Thousands of devices can be monitored by a single sFlow Collector. Assuming that each device contributes 10k flows per second, the sFlow detecting system can handle 10 million flow per second comparing to 30k per second of NOX controller. This approach has a drawback that it adopts the static decision threshold to select elephant flows upon existing solution. However, it is significant to adaptively adjust the threshold according to dynamically changing traffic. Moreover, it involves the problem of low accuracy, which might result in positive false error and negative false error.

5. Host-based detection: Comparing to set up a sniffer or applications at the switch, this approach needs to modify in each host. It can detect elephant flows before transmission immediately and accurately. When the measurements of a flow (eg, socket buffer and flow size) exceed a previously set threshold value, the detector determines that the flow is an elephant flow. Designing the adaptive detecting system has attracted the interest of many researchers, and it is more accurate to use the traffic prediction theory. This method can minimize the overhead without saving the statistics of flows. However, this method is not practical in data center as the operation system of each end host must be changed. Currently, virtualization technology has been widely applied to data center. In this scenario, a single host may run multiple virtual machines that have to be modified and install the detecting application.
3 | SYSTEM DESIGN

An unexpected variation of the traffic pattern in data center networks is common, which will change the definition value of elephant flows. If the detection system decision threshold is statically configured, the detector maintains the same threshold without consideration of the change of network character. The excessively high limitation of threshold will result in high false negatives that a lot of elephant flows are detected. On the contrary, under the low threshold edge switches will generate too many mice flows that cause the routing path congested. It urgently requires an adaptive approach of threshold for elephant flow detection with the rapidly changing traffic.

3.1 | Overview

To address the cause of high detection error rates in data center, we present an adaptive approach of threshold for elephant flow detection that uses DTL algorithm to predict the future traffic and configures the threshold real-timely and adaptively. The design principle is that the mechanism must identify elephant flow with the demands of the traffic characteristics in data center, which needs no extra modification to OpenFlow protocol. The above principle guarantees that our solution is practical with little workload to upgrade. However, there are 3 challenges to resolve: first, how to dynamically meet the demands of the traffic characteristics in data center? Second, how to make sure that the predicted threshold value is accurate? Third, how to prevent the flow entries flapping which caused by configuring the predicted threshold value to switches frequently?

The underlying logic of our solution is as follows: using DTL algorithm to predict the future traffic and decide the threshold value with weighted optimization and smooth mechanism. To address the above challenges, we propose a 2-layer efficient architecture illustrated in Figure 3. The top layer is the module for collecting the traffic status and analyzing the information. The collector learns traffic statistics from every switch periodically and sends them to analyzer immediately. Analyzer gathers the suitable characteristics to DTL algorithm. The second layer, named DTL algorithm, is the core component of system. It consists of 3 parts: training data, weighted optimization, and smooth mechanism. Training data mainly build the mathematical model according to the suitable characteristics and predicts the threshold value with weighted optimization. The benefit of weighted optimization is to ensure the processing of training data accurate. Smooth mechanism decides whether to replace the previous threshold value based on difference estimation or not.

The workflow of detection is as follows:
1. At the beginning, when a new flow comes, the switch decides whether it belongs to elephant flow or mice flow according to the preconfigured threshold value.
2. Traffic statistics is collected from switches periodically and updated in the traffic collector.
3. Analyzer gathers the suitable characteristics from traffic collector and sends data to training data.
4. Training data build the mathematical model on the basis of the suitable characteristics and predicts the threshold value with weighted optimization.
5. Compared with the previous threshold value, smooth mechanism judges the predicted threshold value whether to be configured in the switches for classifying the elephant flow or not.

FIGURE 3  Elephant flow detection architecture
From the above workflow description, it is easy to note that the controller no longer collects the flow statistics for classifying the elephant flow. In the meantime, training data meet the demands of the traffic characteristics by collecting traffic statistics periodically. Weighted optimization makes certain the predicted threshold value accurate and smooth mechanism prevents the flow entries flapping. Therefore, the detection keeps the advantage of flexibility of SDN and reduces the overhead on control plane.

3.2 | DTL algorithm

Machine learning systems automatically learn programs from data. This is often a very attractive alternative to manually construct them, and in the last decade, the use of machine learning has spread rapidly throughout computer science and beyond.24 In this section, we focus on showing how to use machine learning to meet the demands of the traffic characteristics and present our mathematical model in detail.

3.2.1 | Training and optimizing

Let $p_i$ be the traffic statistics of flow sizes in bytes that is collected by traffic collector per second. We define the suitable characteristics during $\Delta t$ seconds in Equation 1.

$$ q_j = \theta \cdot \sum_{\Delta t} p_i $$

If the proportion of flow size of total traffic during $\Delta t$ seconds is greater than factor $\theta$, we regard the flow as an elephant flow. Thus, let $q_j$ be the threshold that is used to indicate the future threshold $Th_k$. To meet the demands of traffic characteristics in data center, we define the predicted threshold value $Th_k$ in Equation 2.

$$ Th_k = \psi (q_{k-1}, q_{k-2}, \ldots, q_{k-N}, \alpha) $$

s.t. $k > N$

Here, $\alpha$ is the cooperative parameter that relates the $Th_k$ with $q_j$. $k$ can be regarded as the current moment. $N$ can be regarded as a moving window to select $q_j$. $\Psi$ is the function for predicting the threshold value. In Equation 2, the key sight is how to find the cooperative parameter $\alpha$. However, it is difficult to solve this problem. But we can regard $\{q_{k-1}, q_{k-2}, \ldots, q_{k-N}\}$ as the data set of $\{(X_n, Y_n)\}$ ($n = k-1, k-2, \ldots, k-N$). Thus, if we can find the solution that $\psi (X, \alpha) \in \Psi$ in the function $\Psi$ to make $E^2$ minimum, the equation of $Th_k = \psi (X_k, \alpha)$ is accepted. The definition of $E^2$ is described in Equation 4.

$$ E^2 = \sum_{N} (\psi (X, \alpha) - Y)^2 $$

Therefore, we can transfer the problem to use $N$ data estimating the cooperative parameter, which minimizes $E^2$. The least squares method is adopted to solve Equation 5 into the following problem:

$$ \alpha^* = \arg \min \alpha \|A\alpha - B\|^2 $$

where

$$ A = \begin{bmatrix} X_0^{k-1} & \cdots & X_M^{k-1} \\ \vdots & \ddots & \vdots \\ X_0^{k-N} & \cdots & X_M^{k-N} \end{bmatrix} $$

$$ B = [q_{k-1}, q_{k-2}, \ldots, q_{k-N}]^T $$

s.t. $N > M$.

$M$ represents the number of order in $\psi (X, \alpha)$. To improve the accuracy of the predicted threshold value $Th_k$, we propose the weighted optimization to optimize the processing of machine learning. We consider that $\omega$ is defined to identify the importance of different time, $\omega_{k-1} > \omega_{k-2} > \ldots > \omega_{k-N}$. In this paper, $\omega$ follows the Gaussian distribution. Thus, the minimization of Equation 5 is translated into the following problem:

$$ \alpha^* = \arg \min \omega \|A\alpha - B\|^2. $$
To reduce the complexity of machine learning algorithm, QR decomposition is used as follows:

\[ A = Q \begin{bmatrix} R_0 \\ 0 \end{bmatrix}, \]  

(10)

where \( Q \in \mathbb{R}^{N \times M} \) is orthogonal matrix and \( R \in \mathbb{R}^{M \times M} \) is upper triangular matrix. Then the optimal solution of Equation 10 is as followed:

\[
\omega \| A \beta - B \|_2^2 = \omega \left\| Q \begin{bmatrix} R_0 \\ 0 \end{bmatrix} \beta - B \right\|_2^2 \\
= \omega \left\| R_0 \beta - Q^T B \right\|_2^2 \\
= \omega \left\| \begin{bmatrix} R_0 \\ 0 \end{bmatrix} \beta - \begin{bmatrix} d \\ e \end{bmatrix} \right\|_2^2 \\
= \omega \| R_0 \beta - d \|_2^2 + \omega \| e \|_2^2
\]  

(11)

Where we define \([d, e]^T = Q^T b \) with \( d \in \mathbb{R}^M \) and \( e \in \mathbb{R}^{N-M} \). Equation 11 reaches minimal if and only if \( R \beta = d \), leaving the second team \( \| e \|_2^2 \) as the residual of the rest squares problem. Therefore, QR factorization simplifies the least squares problem to a linear system with a single unique solution \( \alpha^* \) that is as following:

\[ \alpha^* = R^d. \]  

(12)

According to the above analysis, the DTL algorithm can be constructed by optimal algorithm 2. In algorithm 2, \( Th_{pre} \) means the threshold value in the current configuration and \( q_{pre} \) means the training value gathered by Equation 1. However, the predicted threshold value \( Th_k \) is updated frequently so that the elephant flow detection can meet the demands of traffic characteristic. The result caused by this is that controller configures the switches too many times in a very short time and the flow entries may be flapping because of the difference definition of elephant flow. To prevent the flow entries flapping, we propose a smooth mechanism.

**Algorithm 1 DTL algorithm**

1. function UPDATE_THRESHOLD\((k, Th_{pre}, q_{pre})\)
2. Function Order \(\leftarrow N\)
3. Sampling Length \(\leftarrow M\)
4. if Function Order > Sampling Length then
5. return error;
6. end if
7. if \(k <\) Sampling Length then
8. return \(Th_{pre}\)
9. end if
10. \(Th_k \leftarrow LEASTSQUAREMETHOD(k, q_{pre})\)
11. \(Th_{new} \leftarrow SMOOTHMECHANISM(Th_{pre}, Th_k)\)
12. if \(Th_{new} \neq Th_{pre}\) then
13. return \(Th_{new}\)
14. end if
15. end function
16.
17. function LEASTSQUAREMETHOD \((k, q_{pre})\)
18. \(A \leftarrow\) Obtained by Equation 6
19. \(B \leftarrow q_{pre}(k - Sampling\ Length) : (k - 1)\)
20. \([Q, R] = QR(A \ast \omega)\)
21. \(\alpha = pinv(R) \ast Q^T \ast (\omega \ast B)\)
22. return \(Th_k = \psi(X_k, \alpha)\)
23. end function
3.2.2 | Smoothing

In this section, we mainly discuss the smooth mechanism in detail. As the elephant flow detection needs to meet the demands of the traffic characteristics in data center, the number of period should not be too long. In other words, we should frequently predict the threshold value to identify elephant flow real-timely and adaptively. When the controller configures the switches, a new threshold value, the arrived flow may be regarded as elephant flow and will request controller to compute noncongestion path. After the scheduling in the controller, a new flow entry will be set up in the switch to re-route the elephant flow. The next moment, this elephant flow is identified as mice flow based on the latest threshold value, and the new allocated flow entry will be removed immediately. This case not only causes a lot of false positive errors and false negative errors but makes the flow entries flapped.

To address this issue, we propose the smooth mechanism in the elephant flow detection. When the module receives the predicted threshold value, it will decide whether to replace the previous one based on the difference estimation or not. Thus, we define the amplitude ratio $\beta$ as followed:

$$\beta = \frac{|Th_{new} - Th_{pre}|}{Th_{pre}}.$$  \hspace{1cm} (13)

Let $\varepsilon$ be the smooth parameter and $\tau$ be the smooth time. If $\beta \leq \varepsilon$, the module considers that the predicted threshold value will make the flow entries flapped and prevents the controller configuring the switches. Until the time of $\beta > \varepsilon$ is $\tau$, the predicted threshold value is configured in switches. The smooth mechanism can be constructed by algorithm 2 in detail. [Correction added on 20 November 2017, after first online publication: vertical bars were missing in Equation 13 and have since been added]

Algorithm 2 The smooth mechanism

1: function SMOOTHMECHANISM($Th_{pre}, th_k$)
2: \hspace{0.5cm} $\beta \leftarrow$ Obtained by Equation 13
3: \hspace{0.5cm} if $\beta > \varepsilon$ then
4: \hspace{1cm} count = count + 1
5: \hspace{1cm} if count > $\tau$ then
6: \hspace{1.5cm} initialize count
7: \hspace{1cm} return $th_k$
8: \hspace{1cm} end if
9: \hspace{1cm} return $Th_{pre}$
10: end if
11: return $Th_{pre}$
12: end function

4 | PERFORMANCE EMULATION

In this section, we introduce the system implementation on the basis of an emulated hierarchy topology. Then we evaluate the feasibility and effectiveness of the detection from different aspects.

4.1 | Implementation

We emulate the functional module of the elephant flow detection that is described above in OMnet++, including the clients, the switches, and the controllers that are compatible with SDN. The elephant flow detection is implemented on top of NOX.\textsuperscript{25} Duration and size of TCP flows generated in our performance emulation are according to the real data traces that are collected on January 4, 2016, from 2 routers in a Tier-1 link, which are downloaded from the CAIDA website. Distributed internet traffic generator is used to generate the traffic. One of the most popular data center network topologies fat-tree\textsuperscript{26} is configured in the OMnet++ for performance testing. Analyzer and machine learning are written in C++. Experiment is conducted on a computer with Intel i7-4710MQ 2.5 GHz (4 cores) processor and 8G RAM. Our notation and the assumed values are shown in the Table 1.

4.2 | Experimental results

We first conduct experiments to find out the relationship between error rates and different $M$. The error rate is defined by Equation 14. As shown in Figure 4, when the number of order $M = 1$, the error rate has little influence with moving window
TABLE 1  The parameter of simulation

| Parameter | Description | Value         |
|-----------|-------------|---------------|
| $F_\text{num}$ | Num. of flow | 100,000 flows |
| $T_\text{num}$ | Num. of analytical period | 300 s         |
| $\theta$  | Factor of elephant flow | 0.1%          |
| $\varepsilon$ | Factor of smoothing | 50%           |
| $\tau$   | Num. of max-counter | 3             |
| $D_\text{num}$ | Num. of depth in fat-tree | 8             |

FIGURE 4  The error rate with different $M$

FIGURE 5  The error rate with weighted optimization

and levels off to 17.02%. When the number of order $M \geq 2$, the error rate is changed heavily. From Figure 4, it is easy to see that the error rate is decreased at high speeds when the number of moving window $N$ varies from 1 to 7. However, when $N$ becomes from 7 to 10, error rate is gradually decreased. Compared to $M = 3$ and $M = 4$, the error rate of $M = 2$ is less than 4.71% and 15.34%, respectively, when the number of moving window $N = 7$.

$$\text{error rate} = \frac{Th_{\text{predict}} - Th_{\text{statistic}}}{Th_{\text{statistic}}}$$  \hspace{1cm} (14)

Weighted optimization test is conducted during $M = 2$. As shown in Figure 5, there is no error rate when $N = 1$ because of the constraint condition $M \geq N$. According to Figure 5, it is easy to see that the weighted optimization decreases the error rate effectively. The max difference between the 2 error rates is 6.43% when $N = 4$. Even though at the minimum of difference, the number is also 5.62% when $N = 10$.

Here, we simulate the elephant flow detection on how to meet the demands of the traffic characteristics in data center. On the basis of Figures 4 and 5, it is obviously to see that using weighted optimization with setting $M = 2$ and $N = 7$ can decrease the error rate heavily. As shown in Figure 6, there is not much difference between statistics value and training value. Especially, when the number of analytical period $T = 100$ and $T = 260$, the traffic rapidly changes in network. From Figure 4, we can
see that the error rate is only 13.86%. Compared to training value, the result to meet the demands of traffic in network in fixed value is much worse. However, if the controller configures every training value to switch for identifying elephant flow, it may increase the overhead on control plane. According to Figure 6, it is clear to see that the smoothing value not only meets the demands of traffic in network but also keeps the threshold value invariability correspondingly. Next, we discuss the error rate with different value.

Figure 7 reports that the error rate with training value is the lowest of all. The error rates of smoothing value and fixed value are the same before \( T = 100 \). After that, the error rate that is caused by the fixed value increases rapidly with the traffic changing in network. However, the error rate with smoothing value is still keep less error rate, and there is not much difference between training value and smoothing value. Thus, from Figure 7, we can see that the smoothing value not only meets the demands of traffic in network but also decreases the few of accuracy.

Figure 8 shows the maximum number of pack-in messages to controller with different methods. At the beginning, we have assumed that the number of flows is constant in the network. In Figure 8, it is clear to see that our system can keep the ratio of mice flow and elephant flow invariable with traffic changing. The number of pack-in messages to controller is about 100 through \( T = 0 \) to \( T = 300 \) by using the proposed approach. Other methods, such as Mahout, Deveflow, have a great increase after \( T = 100 \). As OpenFlow needs status of every flow, the number of pack-in messages to controller is top of all methods.

### 4.3 Overhead analysis

In this section, we mainly discuss the overhead caused by our system in the network. To meet the demands of traffic in data center networks, we propose the machine learning algorithm to predict the threshold dynamically. As the extra mechanism in original system, whether our approach increases the workload on control-plane and adds the delay in detection or not is very important. Table 2 shows the number of configuration message to switch-trigger on control-plane. With the rapidly changing traffic, although network administrator operates fixed value sometimes, the configuration message increases still slow. However,
FIGURE 8  Maximum number of pack-in messages to controller

| T  | 1000 | 5000 | 10000 | 15000 |
|----|------|------|-------|-------|
| Fixed value | 1 | 4 | 11 | 17 |
| Training value | 1000 | 5000 | 10000 | 15000 |
| Smoothing value | 9 | 37 | 61 | 103 |

FIGURE 9  The mean of time for an elephant flow detection

the configuration message increases heavily using machine learning algorithm without smoothing mechanism. The overhead caused by this is not negligible. The performance of machine learning algorithm with proposed smoothing mechanism is much better than unmodified one and slightly worse than previous method.

We consider the mean of time to detect an elephant flow in different system. The detection delay includes flow classification delay in detector, data matching delay in data base, and machine learning delay in our system. In this experiment, we compare the delay performance of Deveflow, Mahout, with our system. As can be seen from Figure 9, Mahout only has few milliseconds of delay as the detection is conducted in end host. Deveflow adopts random sampling that takes 10 of milliseconds to identify an elephant flow in collector. In our system, the delay of detection is similar to other sampling approaches on data plane. Machine learning algorithm only takes few milliseconds for meeting the demands of the traffic characteristics on control plane. Our detection system can detect an elephant flow in few 10 milliseconds. The average duration time of elephant flow is about several seconds, where elephant flow detection delay only accounts a small fraction of the total survival time. Thus, our experiments validate the deployment feasibility of our elephant flow detection system.

5 | CONCLUSION

In this paper, we have presented the design, implementation, and evaluation of elephant flow detection, an approach that dynamically changes threshold value to meet the demands of the traffic characteristics in data center networks. We have described the design details of mechanism and implemented it in a prototype to verify its feasibility and effectiveness. We have
presented numerical results from running experiments on the prototype. The results show that the proposed approach is effective in catching the consideration of the rapidly changing traffic in data center networks and the system significantly reduces the overhead of controller. We have also conducted extensive simulations to evaluate the cost in configuring the predicted threshold to switch-trigger and the delay time caused by machine learning. The results show that the delay time it takes is quite small (in the order of ms).

While the present results verified the feasibility and effectiveness of the enhanced mechanism, it is still an open issue that the interaction between the switch and the controller imposes extra latency such as round-trip time and routing path search time when forwarding new flows in SDN. Intuitively, the controller is frequently invoked by the mice flows since the number of mice flows account for a large portion of the total number of flows. Hence, the frequent controller invocation is mainly responsible for the controller performance degradation and thus increases the flow step latency significantly. In the future, we will investigate this and other related issues, incorporate the corresponding approaches into the system, and verify their effectiveness in a real environment.

ACKNOWLEDGMENTS
This work was supported by 973 Program under grant no. 2013CB329100 and the Fundamental Research Funds for the Central Universities (no. 2015JBM008).

REFERENCES
1. Feamster N, Rexford J, Zegura E. The road to SDN: an intellectual history of programmable networks. ACM SIGCOMM Comput Commun Rev. 2014;44(2):87-98.
2. McKeown N, Anderson T, Balakrishnan H. et al. OpenFlow: enabling innovation in campus networks. ACM SIGCOMM Comput Commun Rev. 2008;38(2):69-74.
3. Benson T, Akella A, Maltz DA. Network traffic characteristics of data centers in the wild. In: Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement. New Delhi, India: ACM; 2010:267-280.
4. Kandula S, Sengupta S, Greenberg A. et al. The nature of data center traffic: measurements & analysis. ACM SIGCOMM Conference on Internet Measurement. Barcelona, Spain: ACM; 2009:202-208.
5. Greenberg A, Hamilton JR, Jain N. et al. VL2: a scalable and flexible data center network. ACM SIGCOMM Comput Commun Rev. 2009;39(4):51-62. ACM.
6. Al-Fares M, Radhakrishnan S, Raghavan B. et al. Hedera: dynamic flow scheduling for data center networks. NSDI, 2010;10:19-19.
7. Lin CY, Chen C, Chang JW. et al. Elephant flow detection in data centers using OpenFlow-based hierarchical statistics pulling. In: 2014 IEEE Global Communications Conference. Austin, America: IEEE; 2014:2264-2269.
8. Liu W, Lu W, Liu Z. et al. Identifying elephant flows using a reversible multilayer hashed counting bloom filter. In: High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems & 2012 IEEE 14th International Conference. Liverpool, England: IEEE; 2012:246-253.
9. Curtis AR, Mogul JC, Tournilhes J. et al. DevoFlow: scaling flow management for high-performance networks. ACM SIGCOMM Comput Commun Rev. 2011;41(4):254-265.
10. Bi C, Luo X, Ye T. et al. On precision and scalability of elephant flow detection in data center with SDN. In: 2013 IEEE Globecom Workshops. Atlanta, America: IEEE; 2013:1227-1232.
11. Curtis AR, Kim W, Yalagandula P, Mahout. Low-overhead data center traffic management using end-host-based elephant detection. In: INFOCOM, 2011 Proceedings IEEE. Shanghai, China: IEEE; 2011:1629-1637.
12. Kreutz D, Ramos FMV, Verissimo PE. et al. Software-defined networking: a comprehensive survey. Proc IEEE. 2015;103(1):14-76.
13. Kim H, Feamster N. Improving network management with software defined networking. IEEE Commun Mag. 2013;51(2):114-119.
14. Jain S, Kumar A, Mandal S. et al. B4: Experience with a globally-deployed software defined WAN. ACM SIGCOMM Comput Commun Rev. 2013;43(4):3-14.
15. Jamjoom H, Williams D, Sharma U. Don’t call them middleboxes, call them middlepipes. In: Proceedings of the third workshop on Hot topics in software defined networking. Hong Kong, China: ACM; 2014:19-24.
16. OpenVSwitch. [online] http://http://docs.openvswitch.org/en/latest/.
17. Basile C, Canavese D, Lioy A. et al. Inter-function anomaly analysis for correct SDN/NFV deployment. Int J Network Manage. 2016;26(1):25-43.
18. Hu Y, Wang W, Gong X. et al. Reliability-aware controller placement for software-defined networks. Wireless Commun Over Zigbee Automot Inclination Meas Chin Commun. 2013;11(2):672-675.
19. Congdon PT, Mohapatra P, Farrans M. et al. Simultaneously reducing latency and power consumption in OpenFlow switches. IEEE/ACM Trans Networking. 2014;22(22):1007-1020.
20. Li J, You J, Hong JW. Dynamic control plane management for software-defined networks. Int J Network Manage. 2016;26(2):111-130.
21. Kumar A, Sung M, Xu JJ. et al. Data streaming algorithms for efficient and accurate estimation of flow size distribution. ACM SIGMETRICS Perform Eval Rev. 2004;32(1):177-188. ACM.
22. Wang M, Li B, Li Z. sFlow: towards resource-efficient and agile service federation in service overlay networks. In: Proceedings - International Conference on Distributed Computing Systems. Tokyo, Japan; 2004:628-635.
23. Kumar A, Sung M, Xu JJ, et al. Data streaming algorithms for efficient and accurate estimation of flow size distribution. ACM SIGMETRICS Perform Eval Rev. 2004;32(1):177-188. ACM.
24. Domingos P. A few useful things to know about machine learning. Commun Acm. 2012;55(10):78-87.
25. Gude N, Koponen T, Pettit J, et al. NOX: towards an operating system for networks. ACM Sigcomm Comput Commun Rev. 2008;38(3):105-110.
26. Al-Fares M, Loukissas A, Vahdat A. A scalable, commodity data center network architecture. ACM Sigcomm Comput Commun Rev. 2008;38(4):63-74.

Zehui Liu received his BS degree from Beijing Jiaotong University in 2012. He is currently working toward his PhD degree in the School of Electronic and Information Engineering at Beijing Jiaotong University. His research interests are Smart Identifier Network architecture, ID/Loc Split network architecture, Soft Defined Networking, Information-Centric Networking, etc.

Deyun Gao received BEng and MEng degrees in electrical engineering and a PhD degree in computer science from Tianjin University, China, in 1994, 1999, and 2002, respectively. He spent 1 year as a research associate with the Department of Electronic and Engineering, Hong Kong University of Science and Technology, Kowloon. He then spent 3 years as a research fellow in the School of Computer Engineering, Nanyang Technological University, Singapore. Since 2007, he is on the faculty of Beijing Jiaotong University as an associate professor of School of Electronics and Information Engineering. His research interests are in the area of wireless sensor network and next-generation networks.

Ying Liu received her BS degree from Beijing Jiaotong University in 2000. She received her MS and PhD degrees from Beijing Jiaotong University in 2003 and 2012, respectively. Her research interests include network security, routing protocols, and network management technologies. Her recent research projects include the research on security of ID/Loc Split network, supported by the Fundamental Research Funds for the Central Universities, and the research on Smart Identifier Network architecture, supported by the National Basic Research Program (973 Program).

Hongke Zhang received his MS and PhD degrees in electrical and communication systems from the University of Electronic Science and Technology of China in 1988 and 1992, respectively. From September 1992 to June 1994, he was a postdoctoral research associate at Beijing Jiaotong University, and in July 1994, he became a professor there. He has published more than 100 research papers in areas of communications, computer networks, and information theory. He is now the chief scientist of a National Basic Research Program (973 program).

Chuan Heng Foh received his MSc degree from Monash University, Australia in 1999 and PhD degree from the University of Melbourne, Australia in 2002. After his PhD, he spent 6 months as a Lecturer at Monash University in Australia. In December 2002, he joined Nanyang Technological University, Singapore, as an Assistant Professor until 2012. He is now a Senior Lecturer at the University of Surrey. His research interests include protocol design and performance analysis of various computer networks including wireless local area and mesh networks, mobile ad hoc and sensor networks, 5G networks, and data center networks. He has authored or co-authored over 100 refereed papers in international journals and conferences. He actively participates in IEEE conference and workshop organization, including the International Workshop on Cloud Computing Systems, Networks, and Applications (CCSNA) where he is a steering member. He is an Associate Editor for IEEE Access, IEEE Wireless Communications, International Journal of Communications Systems, and a Guest Editor for various International Journals. Currently, he is the Vice-Chair (Europe/Africa) of IEEE Technical Committee on Green Communications and Computing (TCGCC) and the Chair of Special Interest Group on Green Data Center and Cloud Computing under TCGCC. He is a senior member of IEEE.

How to cite this article: Liu Z, Gao D, Liu Y, Zhang H, Foh CH. An adaptive approach for elephant flow detection with the rapidly changing traffic in data center network. Int J Network Mgmt. 2017;27:e1987. https://doi.org/10.1002/nem.1987