Journal of Internet Services and Applications

RESEARCH Open Access

Identifying elephant flows using dynamic thresholds in programmable IXP networks

Marcus Vinicius Brito da Silva[1,2*], Jonatas Adilson Marques[1], Luciano Paschoal Gaspary[1] and Lisandro Zambenedetti Granville[1]

Abstract

Internet eXchange Points (IXPs) are Internet infrastructures composed of high-performance networks that allow multiple autonomous systems to exchange traffic. Given the challenges of managing the flows that cross an IXP, identifying elephant flows may help improve the quality of services provided to its participants. In this context, we leverage the new flexibility and resources of programmable data planes to identify elephant flows in IXP networks adaptively via the dynamic adjustment of thresholds. Our mechanism uses the information reported by the data plane to monitor network utilization in the control plane, calculating new thresholds based on previous flow sizes and durations percentiles and configuring them back into switches to support the local classification of flows. Thus, the thresholds are updated to make the identification process better aligned with the network behavior. The experimental results show that it is possible to identify and react to elephant flows quickly, less than 0.4ms, and efficiently, with only 98.4KB of data inserted into the network by the mechanism. In addition, the threshold updating mechanism achieved accuracy of up to 90% in our evaluation scenarios.

Keywords: Software-defined networking, Internet exchange points, Network management, Programmable networks

1 Introduction

Internet eXchange Points (IXPs) are high-performance networks that connect autonomous systems (ASes) of the Internet and enable service providers to perform traffic exchange in order to better serve their customers [17]. IXPs perform an essential role on the Internet ecosystem [4], accounting for at least 20% of all traffic exchanged between ASes [8]. As in any network, IXP operators face management challenges every day to get the best out of their networks and meet the users demands. An important problem in IXP management is identifying elephant flows, which are characterized by having traffic size and duration significantly higher than other flows [2, 6, 12, 13]. If not managed properly, elephant flows can exhaust network resources and consequently impact smaller flows that cross paths with them on the IXP infrastructure, thus compromising the overall perceived network Quality of Service (QoS) [18].

A flow is considered an elephant one when both its data volume and duration exceed certain classification thresholds. Existing solutions – such as DevoFlow [10], OpenSample [23], and SDEFIX [17] – present mechanisms for identifying elephant flows using sFlow [20] to estimate flow size and duration, and using SDN/OpenFlow [19] to manage paths in reaction to elephant flow detection. These approaches rely on extracting flow samples from the data plane and analyzing them in the control plane. Whenever a flow exceeds predefined thresholds, it is classified as elephant and subsequently mitigated according to the network operator's policies. One drawback of these approaches is that there is an inherent delay

*Correspondence: mvbsilva@inf.ufrgs.br
[1]Institute of Informatics, Federal University of Rio Grande do Sul, Porto Alegre, Brazil
[2]Federal Institute of Pará, IFPA, Cametá, Brazil

in communication between data and control plane, which delays the identification and mitigation of elephant flows.

In a previous work, and aiming to identify elephant flows in programmable IXP networks more quickly, we proposed IDEAFIX [11, 21], a mechanism that analyzes flow size and duration upon the arrival of each packet at network edge P4-enabled switches [7]. IDEAFIX computes and stores flow meta-data in P4 registers, indexed by hash keys, and compares them to predefined thresholds to classify flows. However, as initially designed, IDEAFIX uses operator-defined, static thresholds to identify elephant flows.

Recent work has also exploited the SDN's global view of network state and traffic to define and configure switches with adequate thresholds to identify heavy hitters. In NHH [16], switches count the number of bytes transmitted by each flow, and when this number exceeds local thresholds, a notification is sent to the control plane to judge the flow behavior according to the current network state. In the control plane, notifications and aggregate traffic statistics are combined to classify the reported flows. Furthermore, the control plane selectively queries switches to request additional traffic accounting and to update their local thresholds considering the overall network behavior. This approach deals with the limitation resulting from the local-view only available to each individual switch. However, NHH does not identify elephant flows, which are legitimate flows in the network that need to be managed, while heavy hitters can be attacks (e.g., volumetric denial-of-service) and need to be mitigated.

In addition to the quantitative definition (volume and duration), an elephant flow can be arbitrarily determined by the network operator according to its own criteria or network policies. For example, in [9, 18], flows are considered elephants when they both contribute to at least 10% of the total traffic volume and are among the 10% longest active flows, i.e., the classification thresholds are defined according to the network utilization. Therefore, in this paper, we introduce a new approach that combines programmable switches and the dynamic adjustment of thresholds to identify elephant flows in IXP networks. Our mechanism uses the information reported by the data plane to monitor network utilization in the control plane and to calculate local classification threshold values. Thus, the classification is done entirely in the data plane, while the control plane tunes identification thresholds to adapt to changes in traffic behavior.

In order to dynamically update thresholds to reflect network traffic behavior, the control plane periodically triggers an update module according to a frequency t configured by the network operator. When the update module is triggered, it retrieves information about flows observed since the last update from a monitoring database. This module uses the retrieved information to calculate new threshold values according to a percentile of the typical flow size and duration observed in the last monitoring window. Thus, the network operator can set the desired percentile according to network utilization and policies. For example, the operator can specify that flows with features higher than the 90th percentile of all flows observed in that window should be classified as elephants. When new thresholds are defined, they are updated in the edge switches by the controller. Therefore, edge switches can locally analyze and identify elephant flows according to the overall network characteristics.

The experimental results show that our P4 prototype was significantly more efficient than the mechanism implemented with OpenFlow, which is based on previous work. The results show that it is possible to identify and react to elephant flows quickly (less than 0.4ms) and efficiently (with only 98.4KB of data inserted into the network by the mechanism) in a scenario where about 50% of the flows could be considered elephants. Moreover, the packet processing time of the evaluated P4 software switch also influences the reaction time. On a hardware switch, packet processing time would be even shorter [15, 25]. Finally, the mechanism is capable of identifying elephants flows with 90% accuracy even when the update frequency is low (e.g., once per minute) and memory resources are scarce.

The remainder of this paper is organized as follows. In Section 2, we discuss related work. Following, in Section 3, we describe our previously proposed mechanism to identify elephant flows in programmable IXP networks. Next, in Section 4, we present our novel mechanism to update thresholds dynamically and identify elephant flows in IXP networks more accurately. In Section 5, we report on the evaluation of our proposal and discuss the obtained results. Finally, in Section 6, we present our main conclusions and perspectives for future work.

2 Related work

In this section, we present work related to our proposal organized into three categories. First, we review the use of SDN/OpenFlow in IXP networks for identifying elephant flows. Second, we present existing work on analyzing network traffic directly in the data plane using P4. Third, we discuss existing work on the dynamic adjustment of thresholds to identify heavy-hitters in programmable networks.

2.1 SDN in IXP and elephant flows identification

The use of SDN in IXPs has been proposed in SDX [14] as a solution to some problems found in IXP networks, such as the inherent issues of Border Gateway Protocol (BGP), Virtual Private Network (VPN) deployment, and wide-area server load balancing [1]. In SDX, the participating ASes run SDN applications on top of a virtual

controller. The generated policies from all participants are combined and deployed in the IXP infrastructure by a centralized controller. However, SDX does not deal explicitly with elephant flows. In contrast, other approaches propose mechanisms to elephant flows identification and mitigation using SDN in IXP networks, as described below.

In DevoFlow [10], the OpenFlow protocol is used to keep track of elephant flows with different monitoring mechanisms, such as packet sampling and port triggering. DevoFlow changes the OpenFlow switches by adding new features and counters to the hardware in order to facilitate the identification of flows in the network. This has the goal of minimizing the time to identify elephant flows and the number of requests to the controller. However, only when a threshold is exceeded, in terms of byte count, the flow is classified as elephant and rerouted to the least congested path between the endpoints.

In OpenSample [23], sFlow is used to perform flow sampling in a solution to manage SDN networks. Then, flow rates are calculated by subtracting TCP sequence numbers from two samples of the same flow and dividing the value by the elapsed time between them. However, the statistics stored inside the switches are not taken into consideration. Each flow needs to be sampled and sent to the control plane to be processed. This delays the identification of elephant flows and requires from the network transmitting a significant amount of sampling data. When a flow is classified as elephant, it is redirected by the SDN controller to another path.

In SDEFIX [17], an identification module is used to classify flows, analyzing the data collected by sFlow according to predefined rules. When size and duration thresholds are exceeded, the flow is classified as elephant and mitigated according to policies written by the network operator. For example, elephant flows can be routed through alternative paths in the IXP infrastructure so as not to affect the small flows that are sharing the same paths in the network. As in the previous approaches, SDEFIX performs elephant flow analysis and identification integrally in the control plane, and a reaction occurs when predefined thresholds are exceeded.

Even though such approaches are successful in identifying and mitigating elephant flows, they require flow samples to be analyzed in the control plane. Besides delaying the analysis, this process also imposes significant overhead on the network because of the additional volume of monitoring data that needs to be transmitted. In contrast, our approach performs elephant flow identification directly in the programmable data plane, analysis is done on a per-packet basis, and less amount of data needs to be sent to the control plane.

2.2 Flows analysis in programmable data planes

In HashPipe [22], a packet processing algorithm, designed for P4 switches, is proposed to identify heavy-hitter flows entirely on the data plane. Heavy-hitters refers to the k largest flows traversing a link in a time window, according to the number of packets or bytes [22], regardless of their duration. In HHH [5], heavy-hitter detection is performed through a hierarchical aggregation of IP addresses (i.e., subnets flow), as shown in Fig. 1. Similar to HashPipe, the information about flows (e.g., 5-tuple) are mapped by hash functions to be identified and parsed from their key. The source and destination IP addresses, source and destination ports, and transport protocol type are used to create the 5-tuple.

These strategies enabled the analysis and monitoring of the flows directly on the programmable data plane. However, they do not consider the identification of elephant flows, which involves not only size but also the duration of the flows. Despite similarities with elephant flows, such as high data volume, heavy-hitters may also represent malicious traffic and must consequently be treated appropriately [6]. In contrast, an elephant flow consists of a legitimate, high-volume, and long-duration flow that needs to be carefully managed [6, 22]. Furthermore, these proposals were not designed for IXP networks, where a delay in identifying these flows can greatly compromise the performance of the services provided by the network [13, 18].

2.3 Flow analysis with adaptive thresholds

Although existing approaches allow the identification of elephant flows in IXP networks (as discussed in Section 2.1), they do not present strategies to dynamically adjust the thresholds. However, in NHH [16], a mechanism is presented to identify heavy hitters in networks using adaptive thresholds. NHH implements an identification method in edge switches to account for the flow volume information and analyze the behavior of these flows according to local thresholds. Each switch identifies the heavy hitters to be reported to the controller using different local thresholds for different monitoring keys, i.e., hash keys. When a flow exceeds the local thresholds for a key, the switch sends a notification to the control plane to report about this flow. In the control plane, notifications are combined to obtain aggregate statistics and identify heavy hitters among the reported flows. The classification occurs in combination with the control plane to avoid making decisions with local switch parameters.

Furthermore, in NHH, the controller performs selective switch queries to additional counts and frequently updates the local thresholds considering the overall network behavior. This strategy is used to reduce the control plane overhead caused by notifications. This deals with the limitation of switches having only a local-view of the

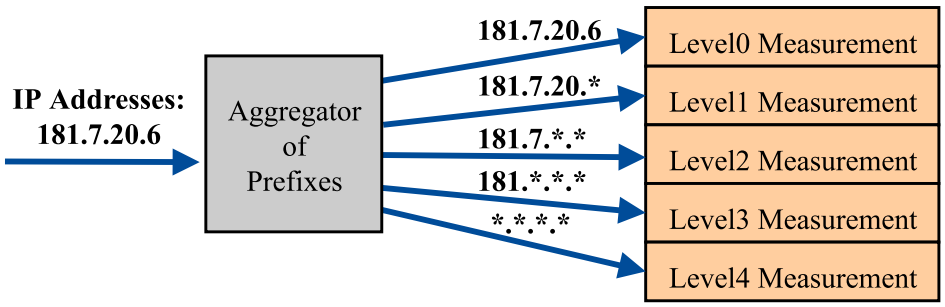

Fig. 1 IP addresses hierarchical aggregation [5]

global network state. Because the controller has a global view, thresholds can be configured locally according to the real network aspects. However, this proposal does not target the identification of elephant flows. In this paper, we present a mechanism to identify elephant flows in programmable IXP networks, based on our previous approach [21], using a dynamically adjusted thresholds.

3 Elephant flows identification in programmable IXP networks

In this section, we discuss aspects of the elephant flow identification process directly in the data plane. We start by briefly describing our initial design [21] for identifying elephant flows in programmable networks. An extension of this design was presented in IDEAFIX [11], which focused on the implementation aspects of the proposed mechanism on P4-programmable devices. In [11], we compared its performance with a traditional identification model based on flow statistic collection with sFlow and traffic management with OpenFlow.

Our elephant identification mechanism computes each flow's volume and duration and stores this information using hash indexing, which maps each flow individually. Thus, each flow can be analyzed by having its information indexed by a characteristic set that defines it. As an example, a tuple consisting of source and destination IP addresses, transport layer protocol, and source and destination application ports can be used to generate an index key. Therefore, for each packet arriving on an edge switch, in addition to traditional routing processing, its volume and duration values are computed and stored in indexed counters with the keys of its flow. In our P4 implementation [11], register externs [7] are used to store these flow features.

To minimize the possibility of information from different flows aggregating into the same counter index – a scenario that may occur when there are collisions in the hash mapping – a multilevel mapping is adopted [24]. The mechanism uses a different hash function for each level so that the indexes a_i for a single flow are different across levels. This mechanism is illustrated in Fig. 2, where k_i is mapped by different hash functions $(h_1, h_2, ...)$. If a collision occurs, it is possible for at least one counter to have the actual values of the desired flow. However, in the more extreme case, where all indexers may collide with indexers of other flows, there is still the possibility of obtaining the value closest to the real flow features value.

When a packet arrives on a switch, at least two keys are generated to index the counters that will store the flow information. Suppose that only one of the keys collides with another flow. As a result, their values are accumulated together in one counter, while the other counter (indexed by the second key) will have a lower (correct) value. In order to reduce the probability of false positives and false negatives, the mechanism always considers the lowest recorded value of each flow when classifying them.

However, to deal with the flows temporal aspect, it is necessary to store the last packet arrival time to obtain the difference with the next packet arrival time. This process allows determining whether a flow is active or has restarted. That is, a timeout can be defined so that only when the difference between the new and the previous packet exceeds this threshold, we may conclude that this is a new flow and reset the counters. This strategy also allows preventing the flow values already stored from being reset in the event of a collision with the new flow keys. If the counters were restarted, the older flow (which is more likely to become an elephant flow) would assume the temporal information of a newer flow. This could cause false negatives, i.e., when one or more elephant flows are not identified.

Furthermore, if both keys of a new flow collide with the keys of other currently active flows, the counters will not be restarted, and the new flow will assume the already counted previous flow values. Thus, we may overestimate flow volume and duration values, but we do not

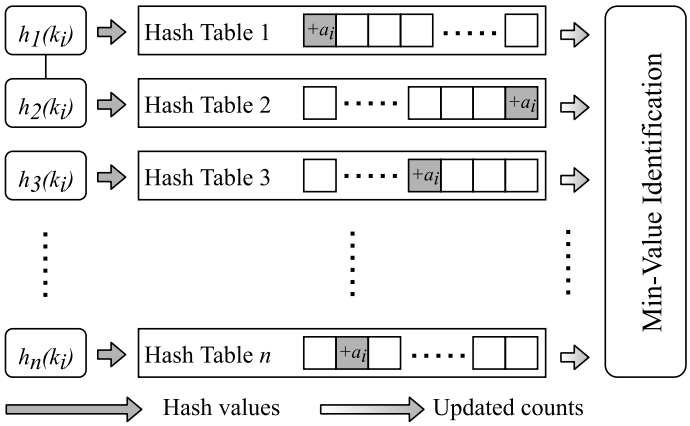

Fig. 2 Multilevel Hash Mapping [24]

underestimate them. That is, it is acceptable that there are false positives (i.e., a flow being classified as an elephant flow when it is not) to the detriment of false negatives, since the interest is in identifying the elephant flows, because these can hurt the smaller flows performance. However, the possibility of multiple key collisions can be minimized by adjusting (increasing) the memory allocated for hash mapping. Although memory limitation is a great challenge in network devices, in this proposal, the dedicated switches memory space to identification mechanism can be adapted to the desired accuracy.

When flow features are defined, they are compared to thresholds that have the role of characterizing flows as elephant or not. These steps are performed entirely in the programmable switches. It is noteworthy that only the edge switches implement the elephant identification mechanism. Network core switches only perform traditional packet forwarding. This design has both the purpose of avoiding imposing overhead on core devices as well as enables flows to be analyzed directly upon their ingress in the network. When a switch classifies a flow as elephant, a notification is sent to the control plane to report the new identification. A packet containing flow features is sent to the controller so that it learns about the identified flow. Thus, actions can be taken in the control plane to manage this flow following the defined network policies. Several elephant flow mitigation policies can be applied; as it is not the focus of this paper to manage elephant flows, we use as an example alternative path routing.

Finally, the thresholds are defined globally and can be updated at run-time by the network operator, allowing for more flexible management. However, dynamically calculating and configuring threshold values was not addressed in previous work. In this paper, we present a mechanism for dynamically updating the thresholds according to the network utilization, as presented in the following section.

4 Approach to dynamic threshold update

This section describes our proposed mechanism to adjust thresholds dynamically to identify elephant flows in programmable IXP networks accurately. The mechanism, illustrated in Fig. 3, uses information from the data plane to monitor the network traffic behavior and calculate and update identification thresholds on the switches, as described in the following sub-sections. Figure 3 shows the architecture of our proposal, composed of an IXP network infrastructure abstraction with a programmable data plane, a historical database, and a threshold update module in the control plane.

4.1 Flow monitoring and database

The update mechanism needs to monitor network traffic behavior. Thus, information about flows is extracted directly from the data plane. This information is computed on edge switches using the identification mechanism presented in Section 3.

However, in the current approach, switches report information for every flow (even those not identified as elephants) as they are finished to build a database. When a flow ends (e.g., TCP FIN Flag is valid or timeout exceeded), a notification (on top of UDP) is sent to the control plane to report the flow size and duration along with its 5-tuple and end flow timestamp. This notification is a cloned control packet from the original packet (e.g., TCP FIN) processed by the switch and sent to the control plane. This strategy reduces the additional management traffic in the network as compared to that generated with a monitoring strategy that performs periodic flow sampling (or packet sampling), as shown in the evaluation, Section 5.

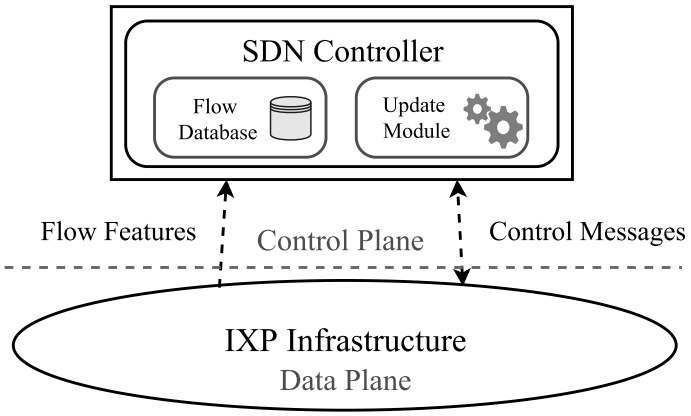

Fig. 3 Architecture of the proposed approach

4.2 New threshold updates

This subsection describes the procedure performed by the update module to define new threshold values. We use a threshold update module to calculate the new threshold values in the control plane from the flow behavior information obtained from the data plane. The network operator can define a t frequency to trigger the update module. Whenever the update module is triggered, the information obtained from the previous monitoring window's flows is retrieved from the database (previous subsection) to be used in the update process, i.e., we use the previous flow's sizes and duration to calculate the new volume and duration thresholds, respectively.

The new threshold values are defined as a percentile of the information obtained from previous flows. We calculate the new thresholds as the P^{th} percentile of volume and duration values of the previous monitoring window's flows. The P^{th} percentile ($0 < P \leq 100$) of a list of N ordered values (sorted in non-decreasing order) is the smallest value in the list such that no more than P percent of the data is strictly less than the value and at least P percent of the data is less than or equal to that value. This is obtained by first calculating the ordinal rank and then taking the value from the ordered list that corresponds to that rank. The ordinal rank n is calculated using the nearest-rank method, following Eq. 1.

$$n = \left\lceil \frac{P}{100} \times N \right\rceil \tag{1}$$

Considering $P = (100 - p)$, the network operator can define which approximate fraction of the network flows observed will be above the thresholds. We define the new thresholds according to the p% largest flows observed in the monitoring window. Thus, the network operator can determine, for example, elephant flows that admit a behavior of at least 10% (i.e., $p = 10$) of the total network behavior observed within that window. This allows operators to determine how conservative the new thresholds will be. For example, in scenarios where the network is saturated, the network operator can be more rigorous and set a lower percentile so that the thresholds identify a greater range of flows as elephants. This allows the thresholds to adapt to the observed network traffic while following operator-defined network policies.

Lastly, when the thresholds are defined, the controller sends control messages to edge switches to configure the new values. Control packets are sent to the switches informing the new threshold, which will be stored in the switch state memories (e.g., in the P4 registers). Thus, edge switches can have thresholds updated dynamically, at run-time, according to the network behavior, and following the network operator parameters. In the following section, we present the main results obtained in the evaluation of the proposed mechanism.

5 Evaluation

In this section, we present our evaluation methodology and discuss the experimental results obtained. In order to evaluate the proposed elephant identification mechanism, another mechanism was developed with the same purpose but implemented with the OpenFlow protocol and based on related work. We create a tool to analyze flows and identify elephants in the IXP network using SDN/OpenFlow.

In the OpenFlow-based mechanism, the information about flows is obtained from the status of the rules installed on the switches. The controller performs requests of type *FlowStatsRequest* to obtain the accounting information for each routing rule installed in a switch. This request is answered with a message of type *Flow-*

StatsReply, containing the duration information, volume in bytes, and packet quantity for each rule. In this paper, the interval between two request-response cycles is called the sampling interval.

Because routing rules are assigned to each flow, it is possible that the information characterizes each of them individually. This feature available in the OpenFlow protocol was used because the counting is performed for each packet in each routing rule. Thus, this mechanism admits a behavior similar to the one developed in P4, which performs the analysis process for each packet passing through the switch.

5.1 Scenario

The test scenario used in the experiments abstracts an IXP network infrastructure, as shown in Fig. 4, based on related work [3, 11, 17, 18]. It is possible to observe that 8 ASes are connected to the IXP network by the edge switches, and these have at least two paths to the network core. In the OpenFlow comparison scenario, similar to the mechanism in P4 (which is implemented only in edge switches), only the edge switches are queried for the routing rules information. This condition is sufficient for the mechanism's operation, since the packets are analyzed as soon as they enter the network.

We generated a workload with distinct sizes of TCP flows between each pair of connected ASes. Flows were generated considering two factors: bandwidth consumption and duration. The flow bandwidth was established at 10 Mbps. The duration of each flow was chosen from a normal distribution. For elephant flows, we used a normal distribution with an average of 150 s and a standard deviation of 20 seconds. In turn, for small flows, we used an average of 10 s and a standard deviation of 4 s [17]. For each evaluation round, 2048 flows were generated, of which approximately 12% were elephant flows. Each experiment lasted 10 min and was repeated 32 times. The elephant flow classification thresholds were initially set to 15s and 15 MB.

When a flow first arrives in the network, the controller establishes a default route using the shortest path algorithm. When an elephant flow is identified, it is redirected by an alternate path. The optimization aspects related to mitigating elephant flows and their paths are beyond the scope of this work. To prioritize time requirements in the IXP networks, in which decisions/actions need to be taken quickly, a proactive approach has been developed in both mechanisms. When a flow is started, the controller inserts the routing rules to both default and alternate paths. When a flow is classified as elephant, its packets are marked to indicate the identification (e.g., set an IP header flag) and forwarded by an alternate path. In the OpenFlow scenario, this proactive strategy has been implemented so that only one routing rule needs to be added by the controller on the switch through which the flow ingresses the network. This approach demands more memory resources for the routing tables. However, this allows reacting to an elephant flow more quickly, as shown in the following results.

We performed the experiments on a computer with Intel Core i7-4790 processor with 8 cores of 3.6 GHz; 16 GB of RAM; and *Linux Ubuntu* 16.04 LTS. The prototype[1] was implemented in the language $P4_{16}$ and using the software switch BMv2[2] as target. The state-of-the-art mechanism was developed with the *RYU SDN Framework* version 4.2, the *OpenFlow* protocol version 1.3, and *Open vSwitch* 2.0.2. The infrastructure was emulated using *Mininet* version 2.3, with a bandwidth of 1 Gbps per link and no propagation delay. The workload was generated with *iPerf* version 3.0.11. The measurements are performed using *tcpdump* and *Wireshark* to traffic analysis.

5.2 Metrics

We compare the P4 proposed mechanism's performance to the mechanism developed with OpenFlow considering the following metrics: (*i*) accuracy, (*ii*) resource utilization, (*iii*) threshold surpassing reaction time, (*iv*) excess data, and (*v*) monitoring data. The mechanism accuracy is the percentage of flows identified as elephants, analyzed in memory space capacity to hash mapping on switches. For all metrics, lower values are better. The resource utilization is the memory and CPU usage by the update module in the control plane. The threshold violation/surpassing reaction time is the interval between the instant of ingress of the packet that exceeds the thresholds and the moment when the first packet is routed through the alternative path. The excess data is the number of bytes transmitted through the default path since the flow has been identified as an elephant before the alternate route starts to be used. The monitoring data is the data inserted into the network by the identification mechanism. In our P4 mechanism, this is the amount (in bytes) of notification messages sent to the controller to report the identified flows. In the OpenFlow mechanism, this is the amount in bytes of sampled flow data sent to the controller.

5.3 Results

The accuracy of the mechanism (Fig. 5) was evaluated by varying the memory space to hash mapping on the switches about the number of flows inserted in the network (i.e., 2048). We compare the dynamic thresholds update mechanism according to the update rate at 60, 120, and 240 s. Besides, we compare with the approach of static thresholds, based on the related work.

[1] https://github.com/mvbsilva/ideafix
[2] https://github.com/p4lang/behavioral-model

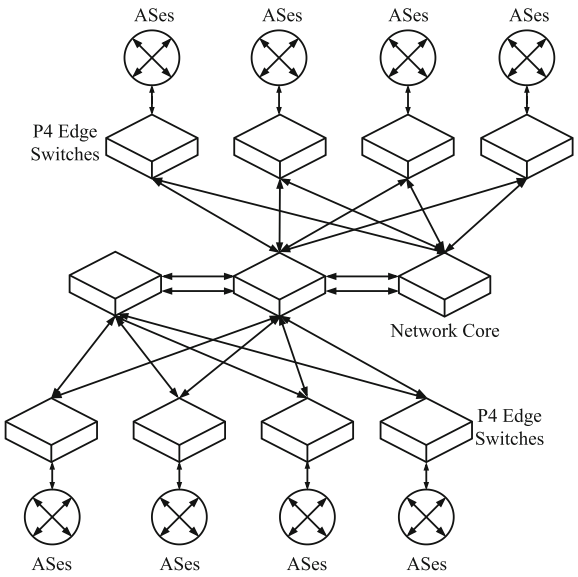

Fig. 4 IXP network topology [17]

With 25% of available memory space, the static threshold approach was able to identify approximately 58% of the elephant flows. With dynamic threshold adjustment, it is possible to increase the identification process accuracy when we reduce the refresh interval. With an update frequency of up to 60 s, it is possible to identify about 90% of the elephant flows, even with scarce memory resources in the switches. This is so because the update module is able to keep up with network behavior faster, with a shorter update interval. Also, in scenarios with higher memory resources in the switches for storing the flow information, about 90% of the elephant flows are identified.

Elephant flows tend to be much larger than other flows in terms of size and duration [8, 9, 13, 18]. The updates frequency variation (interval "t") allows following the rapid variations generated by short flows, observing the overall network behavior, and the network operator's percentile. The flow whose features are strictly below the thresholds behaves closer to elephant flows than others if the percentile is conservative (e.g., 10% [3, 9, 18]). Thus, if an elephant flow is classified, it continues to be treated as an elephant flow even if an update lowers the thresholds.

Table 1 shows the use of resources by the threshold update module. The computational cost, in terms of CPU and memory utilization, were evaluated according to *t* fre-

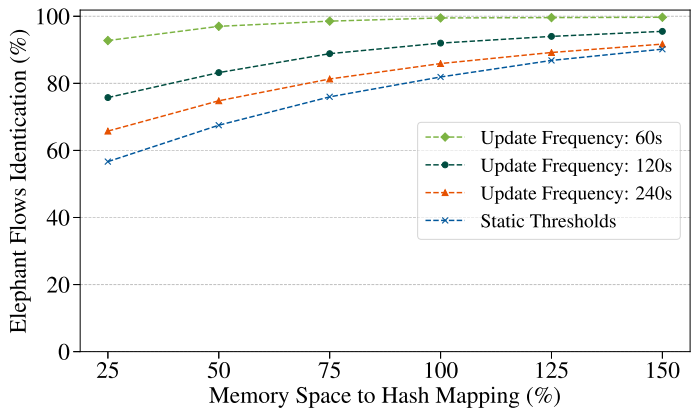

Fig. 5 Comparison between the accuracy of the mechanisms

Table 1 Use of resources by the update module

Update frequency (s)	60	120	240
CPU used	7.3%	18.5%	32.2%
Memory used (MB)	0.2	0.8	1.9

quency to trigger the update module (see Subsection 4.2). When the interval between updates is shorter, approximately 60 s, the computational cost is lower due to the small amount of accumulated information. When the range is higher, about 240 s, more information is accumulated in updating the new thresholds, which incurs more computational cost in the control plane. However, when the interval between updates is shorter, the controller sends more control messages to update the switches' thresholds.

The following results show the performance evaluation of the elephant flow identification mechanism. For comparison, the mechanism developed in OpenFlow was also evaluated with a reactive strategy. In this case, the controller only inserts the alternate route to an elephant flow upon identification. The results obtained with our mechanism implemented in P4 switches are described with the abbreviation "P4". For the mechanism developed with OpenFlow, the abbreviation "OFP" is used for the preventive strategy and "OFR" for the reactive approach.

Figure 6 shows the results for the network reaction time. The x-axis plots the evaluated approaches. We analyze the approaches implemented with OpenFlow with sampling intervals ranging in 0.5, 1, and 5 s. The y-axis presents the average reaction time (in milliseconds) for each approach evaluated on a logarithmic scale. The P4 mechanism can identify and react to an elephant flow within 0.4ms, i.e., the packet processing time of the software-emulated P4 switch.

In comparison, approaches using OpenFlow allow a variation between 92.19 ms, in the best case, and 477.14 ms, in the worst case. Since these approaches depend on the controller involvement, this difference can be pinpointed to the delay in communication between switches in the data plane and the controller server in the control plane. The confidence intervals in the proactive and reactive approaches overlap at their corresponding levels. Consequently, it is not possible to conclude that there is a significant difference at a confidence level of 95%.

Figure 7 presents the volume of data that exceeded the thresholds and continued to be routed through the default path until the reaction occurred. In the P4 mechanism, there is no excess data because the reaction is immediate with the packet processing that characterizes the elephant flow. In contrast, for OpenFlow-based approaches, with a 5s sampling interval, on average 165 KB is routed through the default path and, in the worst case, up to almost 900 MB.

The values illustrated in Fig. 7 are a consequence of the communication time between the OpenFlow controller and switches, in addition to the interval between the monitoring messages. We can observe small confidence intervals overlap between proactive and reactive approaches that are at a 95% confidence level. Thus, it is not possible to conclude a significant difference between them at a confidence level of 95%.

Other significant metric concerns the monitoring data inserted into the network by the approaches, shown in Figs. 8 and 9. Our mechanism only sends a notification informing the identified flow to the control plane, a 96-byte packet. Thus, the size of the monitoring data is directly proportional to the amount of identified elephant flows. In Fig. 8, we present the monitoring data inserted in the network by the P4 mechanism when the percentage

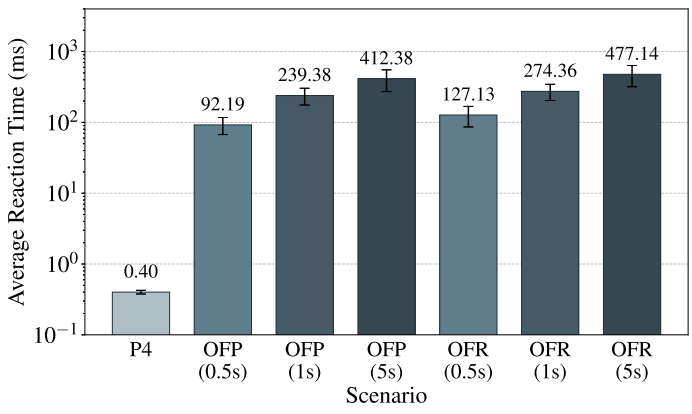

Fig. 6 Threshold violation/surpassing reaction time

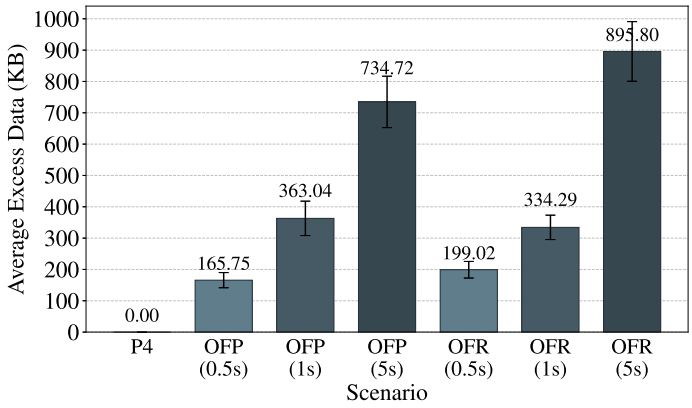

Fig. 7 Excess data

of elephant flows varies between 10, 20, 40, and 50% of the total number of flows.

In the experiments, approximately 2,048 flows were inserted into the network per evaluation round. Thus, in the best case, with 10% elephant flows in the network, about 19.66 KB was inserted as monitoring data. When there were approximately 50% elephant flows in the round, at least 98.31 KB of monitoring data was inserted into the network by the P4 mechanism. In all cases presented in Fig. 8, it is possible to observe that the amount of monitoring data inserted by the P4 mechanism is proportional to the percentage of elephant flows in the network. No significant differences were found considering the confidence intervals with a confidence level of 95%.

From the quantitative analysis of the monitoring data inserted by the P4 mechanism, in Fig. 9, it is possible to observe that there is a significant difference between the P4 mechanism and the approaches implemented with OpenFlow. The scenario with 50% of flows being considered elephant flows allows us to perform an analysis considering more extreme network situations. In OpenFlow approaches, as the sampling interval increases (5s), the monitoring data decreases. However, this implies an increase in reaction time (see Fig. 6), since they are inversely proportional quantities. Finally, there is no difference between the proactive and reactive OpenFlow approaches since the number of messages is proportional to the sampling interval and the experiments' duration.

6 Conclusions and future work

Among the challenges encountered in the management of IXP networks, identifying the elephant flows can contribute enormously to the quality of service provided to the participants. In this way, considering an IXP network with a programmable data plane, this paper presents a mechanism that performs the flow analysis and elephant identification directly in P4 switches, using dynamic thresholds updates.

Our mechanism uses the information reported by the data plane to monitor network utilization in the control plane. The new threshold values are calculated from

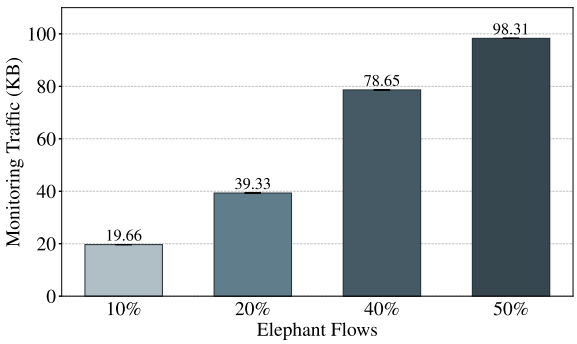

Fig. 8 The P4 mechanism monitoring data

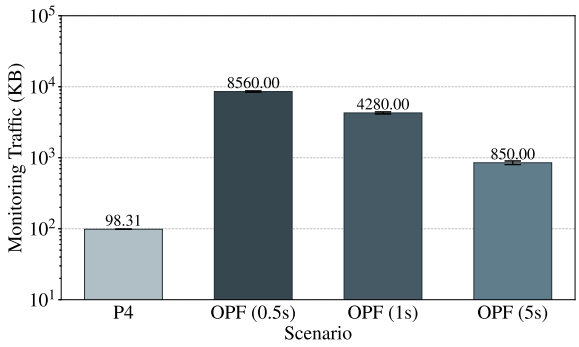

Fig. 9 Monitoring data

the percentile of the previous flow sizes and duration to set the local classification threshold values. Thus, the thresholds are updated to make the identification process more aligned with network traffic behavior. The prototype developed in P4 was significantly more efficient about the mechanism designed with the protocol OpenFlow, in which the process of analysis and identification is carried out with the control plane. In the other hand, on our P4 mechanism, the entire analysis and identification process is performed directly in the data plane. It analyzes each packet arriving on the edge switches and immediately identifies flows that exceed the classification thresholds.

The results demonstrate that it is possible to quickly and efficiently identify and react to these flows by inserting a significantly lower volume of network monitoring data than flow sampling mechanisms. Although the use of memory resources in switches is challenging, the proposal in P4 allows the network administrator to establish the space for use in the mechanism according to the desired accuracy. Finally, in IXPs networks, the elephant flows' identification and reaction must be carried out quickly and accurately. In future work, we will consider other methods based, for example, on machine learning, to calculate the threshold values. We also plan to deploy our solution in a real IXP networks to confirm our findings.

Abbreviations

AS: Autonomous system; IXP: Internet exchange points; QoS: Quality of service; SDN: Software-defined networking; TCP: Transmission control protocol; VPN: Virtual private network

Acknowledgments

Not applicable.

Authors' contributions

M. Silva collaborated in the problem formalization, participated in the proposal of the identification process, participated in the mechanism definition to dynamically update the thresholds, participated in the design of solutions, coded proof-of-concept implementations, ran evaluation experiments, and participated in the analysis of results obtained from experimental evaluations. J. Marques collaborated in the problem formalization, participated in the elephant flows identification proposal, participated in the approach definition to dynamically update the thresholds, and analyzed results obtained from experimental evaluations. L. Gaspary collaborated in the design performance evaluation, and analyzed results obtained from experimental evaluations. L. Granville collaborated in the problem formalization, participated in the design of the elephant flows identification process, and participated in the analysis of results obtained from experimental evaluations. All authors wrote, read, and approved the final manuscript.

Funding

This work was partially funded by the Conselho Nacional de Desenvolvimento Científico e Tecnológico – Brasil (CNPq) by call Universal 01/2016, project NFV Mentor process 423275/2016-0, and Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES).

Availability of data and materials

Please contact authors for data requests.

Competing interests

The authors declare that they have no competing interests.

Received: 22 February 2019 Accepted: 14 November 2020
Published online: 10 December 2020

References

1. Afaq M, Rehman S, Song WC. Visualization of Elephant Flows and QoS Provisioning in SDN-based Networks. In: 17th Asia-Pacific Network Operations and Management Symposium (APNOMS). IEEE; 2015. p. 444–7. https://doi.org/10.1109/apnoms.2015.7275384.
2. Ager B, Chatzis N, Feldmann A, Sarrar N, Uhlig S, Willinger W. Anatomy of a large European IXP. In: ACM SIGCOMM Computer Communication Review, vol. 42. ACM; 2012. p. 163–74. https://doi.org/10.1145/2342356.2342393.
3. Ager B, Chatzis N, Feldmann A, Sarrar N, Uhlig S, Willinger W. Anatomy of a large European IXP. ACM SIGCOMM Conf Internet Meas. 2012;42(4):163–74.
4. Augustin B, Krishnamurthy B, Willinger W. IXPs: Mapped? In: ACM SIGCOMM Conference on Internet Measurement, IMC '09. USA: ACM, NY; 2009. p. 336–49.
5. Basat R, Einziger G, Friedman R, Luizelli M, Waisbard E. Constant Time Updates in Hierarchical Heavy Hitter. In: ACM SIGCOMM Conf Internet Meas. USA: ACM, NY; 2017. p. 127–40.
6. Basat RB, Einziger G, Friedman R, Kassner Y. Optimal elephant flow detection. In: IEEE Conference on Computer Communications (INFOCOM). IEEE; 2017. p. 1–9. https://doi.org/10.1109/infocom.2017.8057216.
7. Bosshart P, Daly D, Gibb G, Izzard M, McKeown N, Rexford J, Schlesinger C, Talayco D, Vahdat A, Varghese G, et al. P4: Programming Protocol-independent Packet Processors. In: ACM SIGCOMM Conference on Internet Measurement. New York: ACM; 2014. p. 87–95.
8. Cardona Restrepo JC, Stanojevic R. IXP Traffic: a Macroscopic View. New York: ACM; 2012. p. 1–8.
9. Curtis AR, Kim W, Yalagandula P. Mahout: Low-overhead Datacenter Traffic Management Using End-host-based Elephant Detection. In: IEEE INFOCOM Conference on Computer Communications. IEEE; 2011. p. 1629–37. https://doi.org/10.1109/infcom.2011.5934956.

10. Curtis AR, Mogul JC, Tourrilhes J, Yalagandula P, Sharma P, Banerjee S. DevoFlow: Scaling Flow Management for High-performance Networks. In: ACM SIGCOMM Conference on Internet Measurement, vol. 41. USA: ACM, NY; 2011. p. 254–65.
11. da Silva MVB, Jacobs AS, Pfitscher RJ, Granville LZ. IDEAFIX: Identifying Elephant Flows in P4-Based IXP Networks. In: Proceedings of the IEEE GLOBECOM Global Telecommunications Conference. IEEE; 2018. https://doi.org/10.1109/glocom.2018.8647685.
12. Gregori E, Improta A, Lenzini L, Orsini C. The Impact of IXPs on the AS-level Topo-logy Structure of the Internet. In: Computer Communications. Elsevier; 2011. p. 68–82. https://doi.org/10.1016/j.comcom.2010.09.002.
13. Guo L, Matta I. The War Between Mice and Elephants. In: Network Protocols, 2001. Ninth International Conference on. IEEE; 2001. p. 180–8. https://doi.org/10.1109/icnp.2001.992898.
14. Gupta A, Vanbever L, Shahbaz M, Donovan SP, Schlinker B, Feamster N, Rexford J, Shenker S, Clark R, Katz-Bassett E. Sdx: A Software Defined Internet Exchange. In: ACM SIGCOMM Conference on Internet Measurement, vol. 44. New York: ACM; 2015. p. 551–62.
15. Hancock D, Van der Merwe J. Hyper4: Using p4 to virtualize the programmable data plane. In: Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies. Irvine: ACM; 2016. p. 35–49.
16. Harrison R, Cai Q, Gupta A, Rexford J. Network-Wide Heavy Hitter Detection with Commodity Switches. In: Proceedings of the Symposium on SDN Research, SOSR '18. New York: ACM; 2018. p. 8:1–8:7.
17. Knob LAD, Esteves RP, Granville LZ, Tarouco LMR. SDEFIX—Identifying Elephant Flows in SDN-based IXP Networks. In: IEEE/IFIP NOMS Network Operations and Management Symposium. IEEE; 2016. p. 19–26. https://doi.org/10.1109/noms.2016.7502792.
18. Knob LAD, Esteves RP, Granville LZ, Tarouco LMR. Mitigating Elephant Flows in SDN-based IXP Networks. In: IEEE ISCC Symposium on Computers and Communication. IEEE; 2017. p. 1352–9. https://doi.org/10.1109/iscc.2017.8024712.
19. McKeown N, Anderson T, Balakrishnan H, Parulkar G, Peterson L, Rexford J, Shenker S, Turner J. OpenFlow: Enabling Innovation in Campus Networks. In: ACM SIGCOMM Conference on Internet Measurement. USA: ACM, NY; 2008. p. 69–74.
20. sFlow. sFlow.org. http://www.sflow.org. Accessed 15 Feb 2019.
21. Silva MVB, Marques JA, Gaspary LP, Granville LZ. Identificaç. In: 36th SBRC Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos; 2018.
22. Sivaraman V, Narayana S, Rottenstreich O, Muthukrishnan S, Rexford J. Heavy-hitter detection entirely in the data plane. In: Symposium on SDN Research. ACM; 2017. p. 164–76. https://doi.org/10.1145/3050220.3063772.
23. Suh J, Kwon TT, Dixon C, Felter W, Carter J. Opensample: A Low-latency, Sampling-based Measurement Platform for Commodity SDN. In: 34th IEEE ICDCS International Conference on Distributed Computing Systems. IEEE; 2014. p. 228–37. https://doi.org/10.1109/icdcs.2014.31.
24. Tong D, Prasanna V. High Throughput Hierarchical Heavy Hitter Detection in Data Streams. In: 22nd IEEE HiPC International Conference on High Performance Computing. IEEE; 2015. p. 224–33. https://doi.org/10.1109/hipc.2015.30.
25. Turkovic B, Kuipers F, van Adrichem N, Langendoen K. Fast network congestion detection and avoidance using P4. In: Proceedings of the 2018 Workshop on Networking for Emerging Applications and Technologies. ACM; 2018. p. 45–51. https://doi.org/10.1145/3229574.3229581.

Publisher's Note