HashCuckoo: Predicting Elephant Flows using Meta-Heuristics in Programmable Data Planes

Marcus Vinicius Brito da Silva*†, Alberto Egon Schaeffer-Filho*, Lisandro Zambenedetti Granville*
*Institute of Informatics – Federal University of Rio Grande do Sul, UFRGS, Porto Alegre, Brazil
†Federal Institute of Pará, IFPA, Cametá, Brazil
Email: {mvbsilva, alberto, granville}@inf.ufrgs.br

***Abstract*—Software-Defined Networking and programmable networks have lead to the development of novel solutions to identify and even predict critical network flows (*i.e.,* flows that can more heavily impact network resources), so they can be properly handled. However, existing approaches found in the state-of-the-art typically incur delays because of the switch-controller communication or depend on thresholds being exceeded to identify flows of interest (*e.g.,* elephant flows). In this paper, we present HashCuckoo, an approach to predict elephant flows that includes: (*i*) a hash-based mechanism to start the prediction process at line rate in P4 switches, based on the Cuckoo Search meta-heuristic; and (*ii*) a local prediction mechanism to infer the new flows' traffic behavior, confirming the classification, and handling elephant flows on-line before exceeding traditionally considered thresholds. We evaluate the trade-offs between HashCuckoo and state-of-the-art solutions, and show that HashCuckoo reduces elephant flow identification delay by 57%, from 102 ms to 43 ms, being the first solution to combine meta-heuristic optimization and prediction that can operate at line rate in programmable data planes.**

***Index Terms*—Network Management, Elephant Flows, Programmable Networks**

I. Introduction

Predicting critical flows that may harm network performance is a challenging task [1]. Among such critical flows, we focus on *elephant flows*, which are characterized by having traffic size and duration significantly higher than other flows (*e.g.,* small and mice flows). Because of their characteristics, elephant flows can substantially impact other flows that share the same network data path. Also, elephant flows tend to rapidly deplete the resources of network devices, which can lead to sub-optimal operation of the network and undesirable delays, queuing, and packet losses [1].

Software-Defined Networking (SDN) [1] and networks with programmable data planes [2] enabled novel solutions to identify and mitigate critical flows using combinations of both centralized and distributed approaches [1]. A number of efforts employ mechanisms to identify elephant flows using, for example, sFlow [3] and managing paths using OpenFlow [4] in the control plane (*i.e.,* at the controller). Others, such as HashPipe [5] and our previous work IDEAFIX [6], identify, respectively, heavy hitters and elephant flows in programmable data planes using P4 [2] (*i.e.,* at the switches). Although P4-based approaches reduce the identification time compared to controller-based approaches (since they minimize the controller-switch communication), thresholds still need to be first exceeded before a flow can be identified as an elephant one (*reactive approaches*). It means that while thresholds are not reached, elephant flows remain unconstrained, thereby impacting network performance [7]. Therefore, a few approaches attempt to predict network traffic behavior and identify flows before exceeding thresholds (*proactive approaches*). This can be done by employing prediction methods to make inferences on the control plane based on statistical models, combining occurrences of previous flows [8]. However, the switch-controller communication can still delay the identification process.

In this paper, we exploit traffic patterns in Internet eXchange Points (IXP) networks [9] and propose a local prediction mechanism to improve elephant flow identification in the IXP programmable data plane. Our proposed mechanism is called HashCuckoo and consists of two collaborative strategies. First, we rely on a hash-based mechanism, optimized by a meta-heuristics called Cuckoo Search Algorithm (CSA) [10], to preliminarily anticipate elephant flows and perform early mitigation in programmable switches at line rate. Second, we implement a local prediction mechanism to infer the new flows' traffic behavior and improve the hash-based mechanism accuracy, confirming/refuting the anticipated identification to handle elephant flows on-line before exceeding traditionally considered thresholds. CSA can solve optimization problems and compose adaptive solutions based on the cuckoo bird behavior. The prediction mechanism is based on the Locally Weighted Regression (LWR) method [11], as used in the state-of-the-art to predict elephant flows in the control plane [8].

Our main contributions with this work are:

(*i*) Propose an approach to predict elephant flows in programmable switches using collaboratively a hash-based mechanism, based on the CSA meta-heuristic, and a local prediction mechanism, based on the LWR method;
(*ii*) Evaluate the performance of our approach and compare our work against two state-of-the-art solutions;
(*iii*) Compare and analyze the trade-offs between our mechanism and previous work to predict elephant flows based on centralized and distributed approaches.

Differently from the state-of-the-art, our mechanism operates entirely in the data plane. To the best of our knowledge, this is the first work to build a prediction mechanism that operates at line rate completely in the data plane, combining meta-heuristics to anticipate elephant flow identification. This enables network operators to apply tailored traffic engineering strategies to handle elephant flows properly, *e.g.,* load balancing, traffic rerouting. The remainder of this paper is organized as follows. In Section II, we detail our proposal to predict elephant flows in programmable data planes, while in Section III we describe our experimental evaluation. In Section IV, we discuss related work. Finally, in Section V, we conclude this paper and discuss future research directions.

II. HashCuckoo

To predict and anticipate elephant flow identification in the programmable networks, we introduce HashCuckoo an approach that consists of two collaborative strategies. First, we use a hash-based mechanism to anticipate elephant flow classification, by analyzing the first packets of a new flow, and mitigation at line rate using P4 switches. Second, we implement a prediction module to infer the traffic behavior of new flows locally in the programmable data plane and confirm or contest the hash-based classification and mitigation actions.

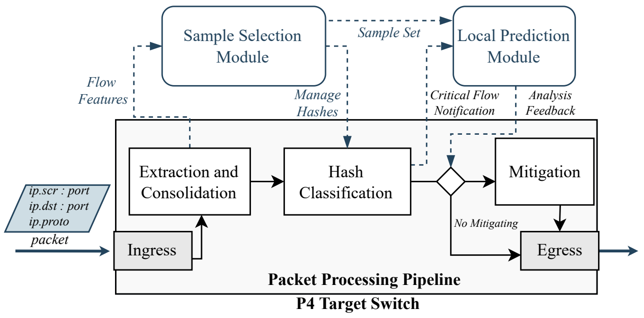


Fig. 1. HashCuckoo architecture in programmable data planes.

Figure 1 illustrates our proposed architecture and workflow. HashCuckoo's *Packet Processing Pipeline* consists of the following steps. First, for each new traffic flow, after the first packet goes through the switch *Ingress*, the *Extraction and Consolidation* step collects and stores in local registers a set of flow data: the flow's *ingressTimeStamp*, packet's length in bytes, and the flow identification 5-tuple (source and destination IP addresses and ports, and transport protocol type). Additionally, the *Sample Selection Module* (described in Section II-A) uses the flow features that are incrementally consolidated (*i.e.,* counting the next packets of the flow) to manage a hash-based mechanism and optimizes the sample set used by the *Local Prediction Module*, as described below.

Next, the *Hash Classification* step checks whether the flow identification key (5-tuple) matches flow instances that were previously identified as elephant ones. If so, the *Mitigation* step applies mitigation actions just before the packet *Egresses* the P4 switch. As a case study, we used alternative forwarding paths to mitigate (isolate) elephant flows. However, different mitigation rules can be adopted according to the policies defined by the network operator.

In parallel, a *notification* is sent to the *Local Prediction Module* to perform a second, more detailed, and sophisticated analysis. When receiving a critical flow *notification*, the *Local Prediction Module* infers the behavior of the new flow from a sample set of the previous flows behavior, using the LWR method (Section II-B). When the inferences are checked based on a prediction interval error tolerance, the values are confronted against local thresholds to characterize the flow as an elephant one or not, before thresholds are effectively exceeded. From there, the analysis feedback will confirm or refute the classification and mitigation actions applied by the hash-based mechanism to minimize false positives.

We used local thresholds in two moments: first, to confront the values inferred by the prediction module (described in Section II-B) and; second, to apply the *filter* rule in the Objective Function (described in Section II-A). However, unlike state-of-the-art reactive identification approaches, HashCuckoo performs identification before thresholds are effectively exceeded.

A. Hash-based Mechanism and Sample Selection Module

HashCuckoo employs a hash-based mechanism to anticipate elephant flow identification in the programmable data plane, using a historical database optimized by the Sample Selection Module (SSM) based on the Cuckoo Search Algorithm (CSA).

Cuckoo Search [10] is a meta-heuristic algorithm to solve optimization problems inspired by the cuckoo bird behavior. Biologically, cuckoo birds do not create their own nests; instead, they lay their eggs in other birds' nests at random. In this case, there is a probability that the host bird will find the intruder egg and throw it away. Thus, the eggs that survive are considered the fittest, and so begins the new *generation*. CSA considers three assumptions: (*i*) each cuckoo bird lays one egg at a time and places it at a randomly chosen nest; (*ii*) the best nests with high quality eggs (solutions $x^{(t)}$, in time t) will lead to the *next generations* (solutions $x^{(t+1)}$); (*iii*) the number of available host nests is fixed, a host can discover an alien egg, and the host bird can either throw the egg away or abandon the nest to build a new nest.

The SSM manages a set of flow instances previously identified as elephant ones (*i.e., Views*). SSM relies on a strategy based on CSA generations and hierarchical IP address aggregation to manage *Views* of previous elephant flow instances stored in the historical database, as illustrated in Figure 2.

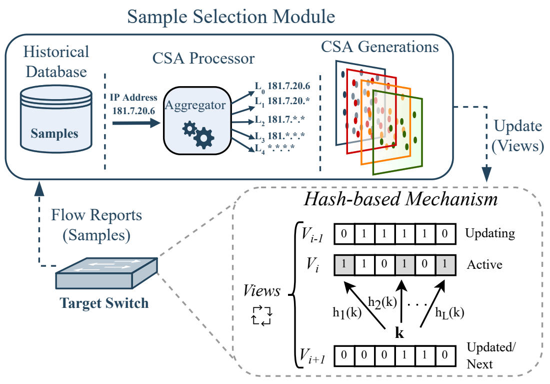


Fig. 2. Hash-based mechanism management and update.

Views are stateful memories (*i.e.,* P4 registers) whose values are read and written by an indexed position. To access indexed positions, we map the flow identification key k by the hash functions h_L, as illustrated in Figure 2. To minimize the possibility of collisions, we adopted a probabilistic structure based on a bloom filter [12] and multiple hash functions. *View* V_i is associated with each hour i of the day (0-23 hours), based on the daily traffic behavior observed in IXP networks [8]. However, alternative *Views* structures can be adapted. When V_i is active, the previous *View* V_{i-1} can be updated. Then, V_i can only be updated when the next *View* V_{i+1} is active.

Initially, we set all filter memory slots of *Views* to zero. The SSM sets to one the indexed slots ($V_i[h_j(k_p)] = 1$) by mapping the identification key k_p of the previous elephant flows that occurred in the network, for L hash functions $h_L(k_p)$. Therefore, when the view V_i is active, if identification key k of the new flow matches ($h_j(k) = h_j(k_p)$) the filter slots of the previous elephant flows: $V_i[h_j(k)] = 1, \forall j \in \mathbb{N} | 0 < j \leq L$; we consider it as a critical elephant flow. Thus, mitigation actions can be applied directly by P4 switches and, in parallel, a *notification* is sent to the local prediction module to perform flow analysis in a subsequent step (described in Section II-B).

We enhance a historical database with flow feature reports (*Samples*) obtained directly from the switch packet processing pipeline, during the Extraction and Consolidation step (Figure 1) to improve the SSM and enable the CSA's generations evolution. *Samples* consist of the flow identification (*i.e.,* 5-tuple), byte count, and first/last packet ingress *timestamp* stored in dedicated registers. Thus, the SSM can create new views and avoid the false positives/negatives in the next generations, using the CSA Objective Function.

The Objective Function has the goal of updating *View* samples from two combined rules: *reduction* and *filter*. In *reduction*, the older samples of a *View* are discarded if they exceed the *time window* defined by the network operator (*e.g.,* seven days). In the *filter* rule, the current *View* samples, and each flow reported as an elephant one within the *View* time interval, have their volume and duration characteristics compared with the (local) thresholds used in that interval. Therefore, only the samples filtered by the objective function will be in this *View* in the next iteration (*next generation*). The worst-case algorithm complexity is given by $O(s + r)$, where s is the number of *samples* in the *View*, and r is the amount of *reported* elephant flows in the *View* active interval.

B. Locally Weighted Regression Prediction Method

To predict network flows behavior, we use a statistical method called Locally Weighted Regression (LWR) [11] [13]. LWR allows to predict the value of dependent variables from a set of independent variables, through localized weighting. LWR assumes that the neighboring values of the desired point in a sample range are the best indicators for the prediction [11]. In this paper, the size and duration of previous flow instances (*samples* in the model) are the dependent variables, their start *timeStamp* are the independent variables, and the new flow size and duration in the start *timeStamp* (desired point) are the predictions of the model. The neighbors are the flow instances with the start *timeStamp* closest to the new flow *timeStamp*.

The LWR method is derived from standard linear regression [11], as shown in Equation 1, and its main idea consists of defining the linear model β parameters, minimizing the sample squared errors $(y_i - f(x_i))$, weighted locally by w_i.

$$F(\beta_0, \beta_1, ..., \beta_m) = \sum_{i=1}^{n} w_i (y_i - f(x_i))^2 \tag{1}$$

The linear model β parameters, used to predict values, are obtained (after derivation [13]) according to Equation 2:

$$\vec{\beta} = \left(X^T W^T W X\right)^{-1} X^T W^T W \vec{y} \tag{2}$$

where X is an $n \times (m+1)$ matrix consisting of n data points, each represented by its m input dimensions and a "1" in the last column, y is a vector of corresponding outputs for each data point, W is a diagonal matrix of order n with the w_i weights of each data point, and β is the $m + 1$ vector of unknown regression parameters. Thus, prediction of the outcome of a query point x_q becomes: $y_q = x_q^T \vec{\beta}$.

The independent variables x_i are the previous flow (*samples*) *timeStamp*, and x_q is the current *timeStamp* (new flow). Observing the temporal correlation and daily periodicity (every 24 hours) in the IXP network traffic behavior [9], we define (in Equation 3) a relative temporal distance d_i between each previous flow and the new flow, assuming that all samples are in the interval $0 \leq d_i < 24$ hours.

$$d_i = |x_q - x_i| \, mod \; 24 = \begin{cases} d_i & \textbf{if } d_i \leq 24/2 \\ 24 - d_i & \textbf{otherwise} \end{cases} \tag{3}$$

The weights w_i allow establishing the influence of the samples according to their distance (*i.e.,* age) to the desired point. Many weighting functions are proposed in the literature [11]. The most widely used weighting function is the Gaussian kernel [13], defined by: $w_i = exp\left(\frac{-d_i^2}{2s^2}\right)$.

Each sample (x_i, y_i) in the model will receive a weight w_i in the interval $[0, 1]$ according to its relative temporal distance d_i to the desired point (x_q, y_q). The s parameter scales the kernel size to determine how local the regression will be. To adjust the magnitude of the Gaussian kernel, we define s according to the *standard deviation* of the *samples*. This allows dynamically adapting the Gaussian kernel according to *variance* in the traffic behavior of the network.

When sample weights are defined, it is possible to determine the $\vec{\beta}$ parameters of the regression model (Equation 2) and then predict the new flow volume and duration y_q, individually, according to its start *timeStamp* (*i.e.,* x_q).

After predicting flows parameters, the mechanism performs the verification of the LWR inferred values. We use *prediction intervals* (Equation 4) [13] to assess the quality of predictions. Prediction intervals I_q are expected bounds of the prediction error at a query point x_q, which can be defined as follows:

$$I_q = x_q^T \vec{\beta} \pm t_{\alpha/2, n'-p'} s^2 \sqrt{1 + x_q^T (X^T W^T W X)^{-1} x_q} \tag{4}$$

where $t_{\alpha/2,n'-p'}$ is Student's t-value of $n' - p'$ (Equation 5) degrees of freedom for a $100(1 - \alpha)\%$ prediction bound.

$$n' = \sum_{i=1}^{n} w_i^2 \quad \text{and} \quad p' = \frac{n'}{n} m \tag{5}$$

Finally, let's suppose that the prediction interval is within the tolerance. In that case, the inferred values are considered acceptable and immediately confronted with thresholds to characterize the flow as an elephant one or not earlier (*i.e.,* before it actually exceeds the thresholds). When the prediction interval is larger than the tolerance, the inference is rejected, as the sample set does not yet have sufficiently correlated instances with the desired flow. The local prediction module can then confirm or refute the classification and mitigation actions applied by the hash-based mechanism.

III. Experimental Evaluation

In this section we evaluate our proposal to predict elephant flows in programmable data planes. First, we outline the experimental setup, the workload characterization and metrics. Then, we divide our evaluation in two steps to elucidate HashCuckoo's two mitigation workflows: firstly, in Section III-D, using immediate mitigation (in which the hash-based mechanism performs the classification when processing the flow's first packet). Next, in Section III-E, when mitigation actions are applied after the local prediction module analysis.

A. Setup

Figure 3 depicts the topology used in the experiments based on the AMS-IX [6]. It presents 8 ASes connected to the IXP network by the programmable edge switches (*i.e.,* P4 switches). Each edge switch implements the HashCuckoo, and the state-of-the-art hash-based and prediction mechanisms.

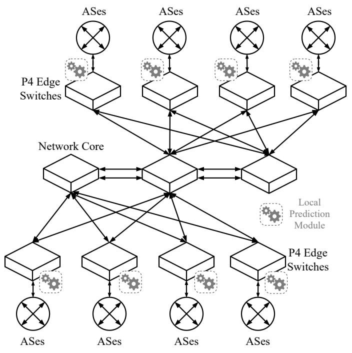


Fig. 3. IXP network topology [6]

We used virtual storage and processing to attach a local prediction module to each P4 edge switch, as shown in Figure 3. We assume that the switches' general-purpose sub-system can execute the local prediction module instructions. In fact, network devices built using ASICs generally include a second general-purpose sub-system (e.g., based on CPUs) to implement the device's monitoring and control functions, as well as more complex and uncommon packet processing functions that the ASIC does not support. Processing in ASICs is usually called the fast path and, by contrast, the slow path is the processing done by the general-purpose sub-system [14].

We performed the experiments on a computer with an Intel Core i7-8565U processor with eight cores of 4.0 GHz; 16 GB of RAM; and *Linux Ubuntu* 16.04 LTS. Our prototype[1] was implemented in $P4_{16}$ using the software switch BMv2[2]. The infrastructure was emulated using *Mininet* version 2.3.

B. Workload

We generated a workload in two scenarios, with distinct sizes of TCP flows between all connected ASes, using *iPerf.* Scenario S1 presents low traffic behavior variation and regular periodicity, *i.e.,* flows follow a well-defined frequency. Scenario S2 presents more variation in terms of flow behavior and periodicity. Flow bandwidth was established at 10 Mbps, and the duration determined through an exponential distribution with an average of 60 seconds, and the rate parameter ($\lambda = 1/\beta$) with $\beta = 20$ (S1) and 40 (S2) seconds, for elephant flows. For small flows, we used an average of 5 seconds and $\beta = 5$ and 10 seconds (for S1 and S2, respectively) [15].

The IXP network traffic was distributed periodically for over nine weeks. Approximately 1,600 hours of previous flows sampling in the network compose the historical databases, distributed proportionally among the eight edge switches and their local prediction modules. The thresholds were defined in 10 MB and 10 seconds [15]. Lastly, experiments were run for 10 minutes, were repeated 32 times, and 8,192 flows were generated per round, of which 12% were elephant flows [9].

C. Metrics

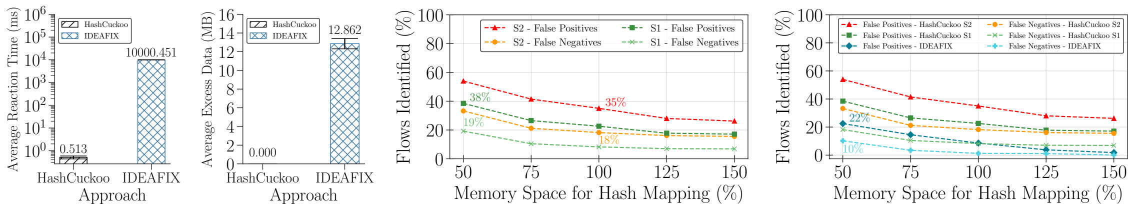
To assess the feasibility of our proposal, we focus on evaluating three main metrics: (i) reaction time, (ii) excess data, and (iii) mechanism accuracy. Reaction time consists of the interval between the arrival time of the first packet at the ingress and the moment when the route of the flow is switched to the alternative path when it is identified as an elephant one. Excess data consists of the number of bytes that went through the default path (until a reaction/mitigation occurs). For accuracy, we consider the percentage of correct elephant flow predictions, false positives, and false negatives. For metrics (i) and (ii), lower values are better.

D. Experimental Results - Immediate Mitigation

We evaluated HashCuckoo against IDEAFIX [6], our previous mechanism to identify elephant flows entirely in programmable data planes. IDEAFIX counts, in hash tables, flow size and duration for each ingress packet and compares them to thresholds to classify elephant flows reactively.

Reaction Time: The results presented in Figure 4(a) show the time required for HashCuckoo to identify beforehand a flow as an elephant one or generate a notification for the local prediction module. We observe in HashCuckoo a 0.51 ms reaction time, while IDEAFIX needed approximately

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



(a) Reaction time. (b) Excess data. (c) HashCuckoo accuracy. (d) HashCuckoo and IDEAFIX accuracy.

Fig. 4. Hash-based mechanisms immediate mitigation: (a) Reaction time using only hash-based approaches; (b) Excess data; (c) HashCuckoo accuracy by varying the hash's memory space using the two workload scenarios; (d) HashCuckoo accuracy, using the two workload scenarios, against IDEAFIX.

10,000.45 ms, considering a 95% confidence level. We emphasize, that for a reaction to occur in IDEAFIX (in the best case), it is necessary to wait for the thresholds to be effectively exceeded (as reported in their paper [6], at least 10 seconds). HashCuckoo reduces elephant flow reaction time as it performs identification before effectively exceeding thresholds.

Excess Data: Figure 4(b) shows the number of bytes forwarded through the default path until the flow is identified as an elephant one and a mitigation policy is applied directly by the P4 switch. In HashCuckoo, there is no excess data because the reaction is immediate after the processing of the flow's first packet by the hash-based mechanism. IDEAFIX admits at least 12.86 MB of traffic forwarded through the default path until thresholds are exceeded and it can perform the appropriate identification and reaction, as mentioned before. These values show that HashCuckoo significantly reduces the impact on flows that compete with elephant flows in the same path.

Accuracy: We evaluated HashCuckoo accuracy by varying the memory space used (on the x-axis, Figure 4(c)) for hash mapping in the switches for a number of flows injected in the network (*i.e.,* 8,192), using the two workload scenarios (see Section III-B). With 50% of available space, the mechanism's accuracy is low since false positives and false negatives achieve values greater than 38% and 19%, respectively, in the best S1 scenario. This is due to the increased hash collisions. When the hash mapping space increases (*e.g.,* 100%), false positives and false negatives decrease (*i.e.,* less than 35% and 18%, respectively, in the worst S2 scenario).

Figure 4(d) illustrates HashCuckoo accuracy, using the two workload scenarios S1 and S2, against IDEAFIX. As expected, IDEAFIX admits greater accuracy (*i.e.,* less than 22% and 10% for false positives and false negatives, respectively, using 50% memory space) than HashCuckoo. This is because IDEAFIX identifies a flow as an elephant only after the thresholds have already been reached, being agnostic to variations in behavior and periodicity of flows. Thus, even though HashCuckoo admits lower accuracy, as was expected for a prediction mechanism, we observed that the reaction time in HashCuckoo is significantly lower, as discussed above.

E. Experimental Results - Prediction Analysis

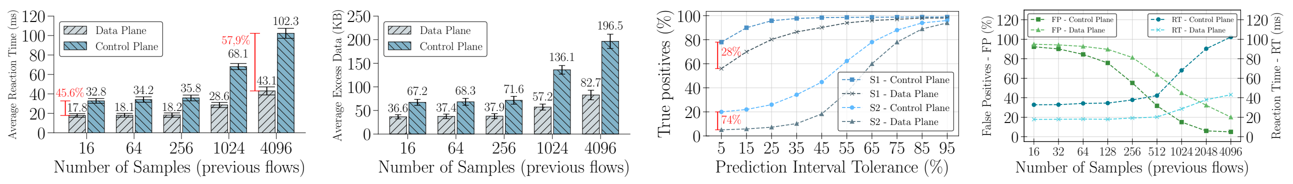
We also compared HashCuckoo against our previous mechanism that performs elephant flow prediction in the control plane [8] to analyze the trade-off between centralized and distributed approaches. This second approach makes similar predictions using the network's global view from the control plane's historical and global database in the SDN controller.

Reaction Time: Figure 5(a) shows the reaction time to elephant flows influenced by the number of samples used in the prediction mechanism. This shows the time needed to react with the prediction model being generated/trained at run-time. The average reaction time (in milliseconds) presents a difference of up to 45% and 57% in the best ($\approx$ 17 ms) and worst ($\approx$ 43 ms) case, respectively, when using the local prediction module, in the programmable data plane, compared to the prediction in the control plane. This difference can be justified by the switch-controller communication latency and the controller database access.

Excess Data: Figure 5(b) shows that the data plane solution reduces the number of bytes forwarded through the default path until the flow is identified as an elephant one, compared to the control plane solution. In the best and worst cases, respectively, the excess data reduces from 82.7 KB and 196.5 KB (the control plane solution) to 36.6 KB and 67.2 KB in the data plane solution, with no overlap in confidence intervals considering a 95% confidence level. These results indicate that our mechanism can reduce by at least 55% the effects of elephant flows on the network QoS.

Accuracy: Figure 5(c) shows the true positives (correct elephant flow predictions) rate for the prediction mechanism based on the prediction interval tolerance. When using workload scenario S1, both prediction mechanisms achieve better results. In scenario S2, the accuracy of the prediction mechanisms is affected, since the prediction interval is sensitive to the sample set correlation. The results show a variation of approximately 28% and 74% between the two approaches, using conservative tolerance for the prediction interval in both scenarios, considering a 90% confidence level. It shows that the method can predict values even in non-regular scenarios. However, it requires more flexibility in prediction tolerance.

Figure 5(d) shows the trade-off between prediction mechanisms on the data and control plane, using S2 scenario and a prediction interval within a 15% tolerance. Our results show, at least, 80% of success in elephant flows identification, out of the total number of elephant flows injected ($\approx$ 983 flows), with 20% of false positives and 8% of false negatives, considering a 90% confidence level, and using 4,096 samples in the prediction model (best-case).



(a) Reaction time. (b) Excess data. (c) True positive predictions. (d) Trade-offs.

Fig. 5. Prediction mechanisms experimental results: (a) Reaction time e (b) Excess data, using data and control plane approaches; (c) True positives rate of the prediction mechanisms; (d) False positives and reaction time trade-off between prediction mechanisms on the data and control plane.

IV. Related Work

In DevoFlow [16] an SDN/OpenFlow-based [4] identification module is used to identify elephant flows, analyzing the collected data by sFlow [3] according to predefined rules. However, these approaches perform elephant flow analysis and identification entirely in the control plane and a reaction occurs only when thresholds are exceeded. IDEAFIX [6] presents an attempt to identify elephant flows in programmable data planes taking advantage of P4 switches to store and analyze the flows' size and duration. Although IDEAFIX reduces the detection delay when compared to controller-based approaches, it still requires flow size and duration to reach certain thresholds for identification to happen. In our solution, we implement a local prediction mechanism to infer the traffic behavior of new flows and identify the elephant ones, before reaching thresholds.

Traffic behavior prediction strategies are alternatives to the approaches described above to identify elephant flows. A number of research efforts aim to predict the behavior of network flows, using Artificial Neural Networks, and Machine Learning [15]. To anticipate elephant flow identification, the authors in [8] adopt a prediction mechanism to infer the traffic behavior in the control plane. However, switch-controller communication causes delay in the identification process [5].

V. Conclusions

In this paper, we capitalized on advanced features of P4 switches to build a mechanism that anticipates the identification of elephant flows directly in the programmable data plane. Experimental results showed that our proposed mechanism, called HashCuckoo, significantly reduces (approx. 57%, in the worst case) the identification time of elephant flows because we eliminate the switch-controller latency in the identification loop. We also evaluated the trade-offs between HashCuckoo and two state-of-the-art solutions to identify elephant flows to provide the network operator with insights about the prioritization between identification time and the mechanism's accuracy. As future work, we will consider other methods based on machine learning to improve HashCuckoo's predictions.

Acknowledgement

This work was supported in part by the Brazilian National Council for Scientific and Technological Development (CNPq) procs. 423275/2016-0 (Universal), 316662/2021-6 (PQ), and 311276/2021-0, by FAPESP procs. 2020/05152-7 (PROFISSA) and 2015/24494-8 (BigCloud), by FAPERGS procs. 16/2551-0000488-9 (Green Cloud) and 19/2551-0001645-0, and by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

References

[1] S. Kaur, K. Kumar, and N. Aggarwal, "A review on P4-Programmable data planes: Architecture, research efforts, and future directions," *Computer Communications*, vol. 170, pp. 109–129, 2021.

[2] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming protocol-independent packet processors," in *ACM SIGCOMM Computer Communication Review*. ACM, 2014, pp. 87–95.

[3] sFlow. sFlow.org. [Online]. Available: http://www.sflow.org

[4] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," in *ACM SIGCOMM Computer Communication Review*. NY, USA: ACM, 2008, pp. 69–74.

[5] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford, "Heavy-hitter detection entirely in the data plane," in *Symposium on SDN Research*. ACM, 2017, pp. 164–176.

[6] M. V. B. da Silva, A. S. Jacobs, R. J. Pfitscher, and L. Z. Granville, "IDEAFIX: Identifying Elephant Flows in P4-based IXP Networks," in *IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2018.

[7] R. B. Basat, G. Einziger, R. Friedman, and Y. Kassner, "Optimal elephant flow detection," in *IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2017, pp. 1–9.

[8] M. V. B. da Silva, A. S. Jacobs, R. J. Pfitscher, and L. Z. Granville, "Predicting Elephant Flows in Internet Exchange Point Programmable Networks," in *International Conference on Advanced Information Networking and Applications*. Springer, 2019, pp. 485–497.

[9] A. Gupta, L. Vanbever, M. Shahbaz, S. P. Donovan, B. Schlinker, N. Feamster, J. Rexford, S. Shenker, R. Clark, and E. Katz-Bassett, "Sdx: A software defined internet exchange," in *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4. ACM, 2015, pp. 551–562.

[10] X.-S. Yang and S. Deb, "Cuckoo search via lévy flights," in *2009 World Congress on Nature & Biologically Inspired Computing (NaBIC)*. IEEE, 2009, pp. 210–214.

[11] W. S. Cleveland and S. J. Devlin, "Locally weighted regression: an approach to regression analysis by local fitting," in *Journal of the American statistical association*, 1988, pp. 596–610.

[12] S. Xiong, Q. Cao, and W. Si, "Adaptive Path Tracing with Programmable Bloom Filters in Software-Defined Networks," in *IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2019, pp. 496–504.

[13] S. Schaal and C. G. Atkeson, "Robot juggling: implementation of memory-based learning," *IEEE Control Systems*, pp. 57–71, 1994.

[14] O. Michel, R. Bifulco, G. Retvari, and S. Schmid, "The programmable data plane: abstractions, architectures, algorithms, and applications," *ACM Computing Surveys (CSUR)*, vol. 54, no. 4, pp. 1–36, 2021.

[15] Y. Li, H. Liu, W. Yang, D. Hu, X. Wang, and W. Xu, "Predicting inter-data-center network traffic using elephant flow and sublink information," in *IEEE Transactions on Network and Service Management*, vol. 13, no. 4. IEEE, 2016, pp. 782–792.

[16] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling flow management for high-performance networks," in *ACM SIGCOMM Computer Communication Review*, vol. 41, 2011, pp. 254–265.