A Hybrid P4/NFV Architecture for Cloud Gaming Traffic Detection with Unsupervised ML

Joël Roman Ky, Philippe Graff, Bertrand Mathieu, Thibault Cholez

A Hybrid P4/NFV Architecture for Cloud Gaming Traffic Detection with Unsupervised ML

Joël Roman Ky*, Philippe Graff†, Bertrand Mathieu*, Thibault Cholez†
*Orange Innovation, Lannion, France, {joelroman.ky, bertrand2.mathieu}@orange.com
†Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France, {first.last}@loria.fr

***Abstract*—Low-latency (LL) applications, such as the increasingly popular cloud gaming (CG) services, have stringent latency requirements. Recent network technologies such as L4S (Low Latency Low Loss Scalable throughput) propose to optimize the transport of LL traffic and require efficient ways to identify it. A previous work proposed a supervised machine learning model to identify CG traffic but it suffers from limited processing rate due to a pure software approach and a lack of generalization. In this paper, we propose a hybrid P4/NFV architecture, where a hardware Tofino based P4 implementation of the feature extraction functionality is deployed in the data plane and a unsupervised model is used to improve classification results. Our solution has a better processing rate while maintaining an excellent identification accuracy thanks to model adaptations to cope with P4 limitations and can be deployed at ISP level to reliably identify the CG traffic at line rate.**

***Keywords*—P4, Traffic Detection, Cloud Gaming, Machine Learning, Virtualized Network Function**

I. Introduction

Cloud Gaming (CG) traffic generated by platforms such as GeForceNow, Microsoft Xbox Cloud Gaming, Sony PlayStation or Amazon Luna is foreseen to take an increasing and significant part of traffic shares in the upcoming years. However, CG traffic is particularly demanding because it requires at the same time a high downstream bandwidth to transmit the high-quality multimedia flow and a very low latency with no jitter to ensure a reactive response to the players' inputs. Previous studies [1] have shown that end-to-end mechanisms are not always sufficient to maintain a satisfactory QoS in some difficult network conditions.

Meanwhile, the recent network technology of Active Queue Management (AQM) L4S (Low Latency Low Loss Scalable throughput) has been introduced and prioritizes the transport of low-latency (LL) traffic using a dedicated queue. However, identifying which flows should benefit from the LL queue remains a challenge. Addressing this issue, our previous work [2] proposed an initial set of VNF (Virtualized Network Functions) able to identify CG traffic at the network edge through a supervised Machine Learning (ML) model based on Decision Trees (DT) using statistical flow features.

In this paper, we propose to extend this work in two important directions that make it even closer to a realistic deployment:

- We propose a new model based on unsupervised ML that shows a far better capacity to recognize traffic from CG platforms that are missing in the learning dataset;
- We translated the VNF facing the traffic to extract the features in P4 (Programming Protocol-independent Packet Processors) and deployed the module on a hardware Tofino based switch.

In addition to those two contributions, we provide an extensive feedback on the adaptations needed to cope with the inherent limitations of P4 programming for hardware switches that enable its guaranteed high bitrate processing.

The rest of this paper is organized as follows. Section II presents the related work on traffic detection and classification using P4. Then, Section III describes our dataset, our previous supervised ML model and the new unsupervised model we propose. Section IV presents our hybrid classification architecture mixing pure software network functions with a new P4 module for feature extraction at line rate, with a particular focus on the limitations imposed by the latter and the required adaptations. Finally, we evaluate the performance of our architecture in Section V before concluding this study in Section VI.

II. Related Work

The Software Defined Networking (SDN) paradigm [3] enables to decouple the control plane from the forwarding plane of network devices. This separation allows their dynamic configuration using interfaces such as OpenFlow [4]. A recent development, based on P4 [5], offers to go further and enables program deployment into the devices in addition to configuration. Several studies investigate the use of P4 for detection of heavy network flows (which may interfere with smaller flows sharing the same path and thus impact the QoS) or for traffic detection. [6] addresses the detection of large flows with an algorithm consisting of a hash table pipeline, which identifies 95% of the heaviest flows on an ISP trace. Other works show threshold-based techniques, either by distributing thresholding across all the switches [7] or by focusing on their size and duration [8]. Some works focus on flow classification using ML models on P4 switches. Inter-arrival times (IATs) histograms are used by [9] to determine which TCP flavour (Cubic or BBR) controls each flow and to classify each flow on P4 hardware using a ML model on the control plane. [10], [11] directly implement ML models on the programmable switches to handle traffic classification. [10] maps four trained

ML models (DT, K-means, SVM, and Naïve Bayes) to match-action pipelines and achieve full-line rate classification. [11] proposes *SMASH-D1* (resp. *SMASH-D2*) to reduce the DT model complexity for flow classification by performing the classification decision one (resp. two) level(s) above the DT leaf nodes.

However, many research papers implement their solution with the P4 Linux software switch (BMv2), and very few on a real P4 hardware switch. This is a crucial point since the software implementation is largely less restrictive than the hardware one. Indeed, a hardware switch must process packets at line rate and has less computational facilities (e.g. no floating-point operations and limited memory-accesses.) than a Linux system. [10] outlines that by adapting their approach to run on hardware-based prototypes, classifiers can use fewer features and classes. [12] investigated to what extent a P4 hardware solution for AQM can be achievable and faced limitations imposed by the P4 hardware. They then split the processing between the P4 data plane for simple operations and control plane for complex computations.

Our proposal differs from the related work in several ways: (i) unlike [6], [8], [11], we perform the traffic detection using a P4 hardware implementation instead of a software one; (ii) we split the CG traffic detection task by performing the features extraction in the data plane with P4 switches and the CG traffic detection in the control plane with a NFV node; (iii) unlike the previous works [2], we use unsupervised learning for CG traffic detection to cope with the generalization problem we may face with supervised learning [13].

III. Datasets and ML Models

A. Datasets collection

We build our CG dataset by collecting network packets with Wireshark (*v3.6.2*). In a first campaign conducted in 2022, we considered the four main CG platforms available in Europe at that time: Nvidia GeForce Now (GFN), PlayStation Now (PSN), Google Stadia (STD) and Microsoft Xbox Cloud Gaming (XC). In a second dataset built in 2023, we collected data on GFN, PSN and XC in addition to Moonlight and Steam Remote Play (STD was shut down by Google in January 2023 but we keep STD data for comparison). The captures result in `pcap` files, all made available for the sake of reproducilibity[1], from which we extract features.

Marchal et al. [1] showed that most CG platforms adapt their traffic to network conditions. Typically, during congestion, CG platforms reduce their bitrate by increasing packet inter-arrival times (IATs) and/or decreasing packet sizes. To take these traffic variations into account, we also captured CG traffic under network constraints. We add a router between the gateway and the CG client to alter the network conditions thanks to *iproute* `tc` commands, and *Wondershaper* for bandwidth limitation (Fig. 1). The generated network perturbations are applied in turn: (i) 20ms of additional delays; (ii) 5 ms of additional jitter; (iii) 5% of packet loss; (iv) 10Mbps of available bandwidth. The host running the CG client performs the network capture.

[1]https://cloud-gaming-traces.lhs.loria.fr/data.html

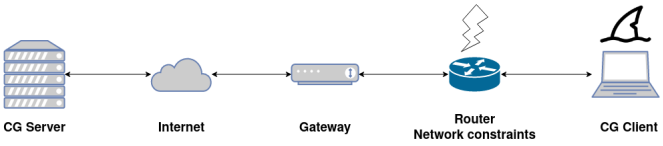

Fig. 1: Testbed for live network packets capture

We also collect Non-Cloud Gaming (NCG) traffic by considering the following high-bandwidth traffic types over UDP; video streaming (VS), live video streaming (LV), video conferencing (VC), and Facebook navigation (FB) over QUIC.

We can then extract features from packet captures. Due to the asymmetry of CG traffic, we differentiate downlink and uplink features. For each direction, we extract six features per time window (we set its duration to 33 ms according to [2]): (1) the total number of packets, (2) the sum, (3) the mean and (4) the standard deviation of packet size, (5) the mean and (6) the standard deviation of packet inter-packet-arrival times (IAT).

B. ML Models

We formalize the CG traffic identification problem as a binary classification problem as follows. Given a dataset of traffic features $x_T = \{x_1, x_2, ..., x_n\}$, we train a model $\mathcal{M}$ which must assign for each new window of features observed $\tilde{x_i}$, a label among two classes within the set {CG; NCG}.

1) DT Model limitations: In our previous works [2], we performed the identification of CG traffic using a *Decision Tree* (DT) model which is a supervised machine learning approach. The goal of supervised learning is to learn the underlying mapping between a new observed instance $\tilde{x_i}$ and its respective label $\tilde{y_i}$, given a set of training examples $\{(x_1, y_1), ...(x_n, y_n)\}$, where x_i and y_i denote the feature vector and its label respectively. We trained the DT model using an equal proportion of CG and NCG instances. The resulting DT model showed good performance for identifying CG traffic even when it was collected under downgraded network conditions. The DT model also classified efficiently NCG traffic types that were present in the training set. To assess the reliability of this model in an open world, we made a new dataset of unseen games executed on previously trained platforms as well as unseen CG platforms. Unfortunately, we realized that, when considering CG traffic not seen during training, the model performance declines (see Fig. 2, blue columns) to unacceptable proportions in the case of new platforms.

2) USAD Model: To bypass the lack of generalization of our supervised model, we propose to identify CG traffic in an unsupervised manner. We use the USAD model (UnSupervised Anomaly Detection) [14], specifically designed for *anomaly detection*, which employs a neural network architecture consisting of two auto-encoders: the model learns, based only on CG traffic features (without the label information), how to

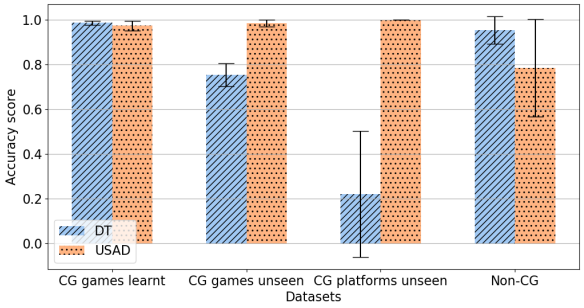

Fig. 2: Performance of DT and USAD models

reconstruct the input data and compute a score s measuring how efficient the model is in the reconstruction task. Since the model has learnt how to reconstruct only CG traffic features, it cannot perform an equivalent reconstruction of the NCG traffic. Hence, the NCG traffic will have a high reconstruction score while CG traffic will have a low reconstruction score. Then, given the reconstruction score s computed by the model considering the features extracted from a given window $\tilde{x_i}$, and a carefully chosen threshold δ, the window is classified as follows:

$$\tilde{y_t} = \begin{cases} CG, & \text{if } s(\tilde{x_t}) \leq \delta \\ NCG, & \text{otherwise.} \end{cases} \quad (1)$$

The rationale behind this strategy is to discriminate the CG traffic from the NCG traffic by relying not on a ground truth to learn discriminating rules or relations but by using the latent characteristics of the dataset [15]. Therefore, the model should be more robust and more generalizable to unseen types of CG traffic.

We split our CG dataset into a train and a test set as follows: the former is composed of normal CG instances from the 4 main CG platforms at our disposal (STD, XC, GFN and PSN). The test sets are made of previous traces from [2]: (i) normal CG instances collected on the same 4 CG platforms, (ii) network constrained CG instances, (iii) NCG instances; and of new collected traces: (iv) CG instances composed of new games captured on XC, GFN and PSN and (v) CG instances collected on new platforms (Moonlight and Steam). The model is then trained on the normal CG dataset only and is evaluated separately on the 5 different types of test set.

The USAD model is used with its default hyper-parameters and is trained with the Adam optimizer with a learning rate of 10^{-3} and a batch size of 128 during 100 epochs. We apply early stopping to avoid overfitting. We compare the accuracy of the two models in Fig. 2, which highlights the superior performance of the USAD model to identify the unknown. More performance evaluation of the model is reported using *Accuracy* and *F1-score* in Section V, for each test set. Our results are not biased because each of the 5 types of test data contains only one of the two classes (CG or NCG).

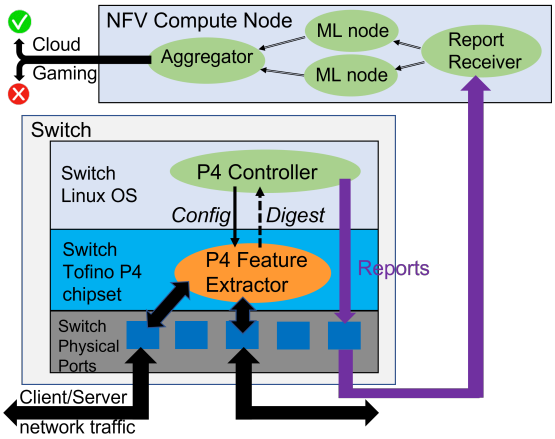

Fig. 3: The 2-level programmable architecture

IV. Architecture and Implementation

Our previous work [2], all software based, can not be deployed elsewhere than at the edge because of its limited capacity to manage more than 30Gb/s on the fly, which prevents for instance a more convenient regional deployment. To overcome this limitation, we propose to split our system into a 2-level *hybrid* architecture: a data-plane programmable module to cope with high speed line-rate traffic to extract flows' features and a control-plane programmable module to perform the complex computational classification functions. For the data plane, we select P4[2] [5], which seems to be the most promising solution for data plane network programmability, and for the control plane, we rely on the NFV architecture.

Fig. 3 depicts our architecture, with a P4-module deployed in the chipset of a hardware switch, connected to a local controller running in the Linux OS of the switch, and 3 computational modules for the detection of CG traffic.

A. The NFV computational modules

Although the hardware P4 module is the primary emphasis of this paper, computational VNFs are introduced below for comprehension. The complex operational tasks, mainly related to the ML processing, as described in Section III are developed in Python3 and run in a Linux environment in user space with all the necessary libraries (pandas, numpy, pytorch, etc.). Since the processing time is long in this environment (at least too long for line-rate processing of high speed networks), we send to them not every packet, but reports, every 33 ms, that include traffic features about the flows transiting via the P4 module during this time window. Furthermore, as presented in Section III, we choose a Deep Learning model USAD, which has very low inference time compared to other solutions.

In this NFV node, 3 types of VNFs are deployed: (i) one for receiving reports, extracting features values and forwarding them to the ML node, (ii) one performing the classification task, for analysing the features and giving a score indicating

[2]the P4 consortium: p4.org

whether this session is cloud gaming or not, (iii) one aggregating the results from the ML nodes per flow and giving a final decision based on several time windows. In this paper, the 3 VNFs are located on the same NFV node, connected to each other via local sockets, but they can easily be deployed on different nodes if required.

B. The P4 extracting module

To achieve a high speed line-rate processing module, we rely on the P4 approach. Since our goal is to have a realistic solution, being potentially deployed in an ISP network, we opt for a P4 hardware switch, and not the Linux-based BMv2 solution. For our testbed, we used a Edgecore Wedge 100BF-32X switch[3], which is equipped with an Intel Tofino P4 chipset. This choice was primarily driven by our performance requirement of several Tb/s but has heavy implications on our work. Indeed, the P4 Tofino architecture differs from the BMv2 one and developing programs for real hardware switches presents many limitations and constraints not present in a Linux-based system (see Section IV-C). This is a crucial factor to consider when compared to previous studies implementing P4 solutions using BMv2.

The Edgecore switch can be schematised with 2 levels: the low-level with the chipset responsible for line-rate packet processing and the high-level, the Linux Switch OS, in charge of the deployment, management, configuration of the chipset. In our implementation, both levels are used. Specifically, the P4 module deployed within the chipset, extracts and computes the basic packet features such as packet size and IAT. Concurrently, the Python-based controller module which runs in the Switch OS, is responsible for configuring the P4 program, receiving the reports from the P4 module (sent using the *Digest* function provided by the P4 Tofino.) and transmitting the reports to the NFV node in the expected format.

C. Limitations of the P4 Tofino hardware solution

In this section, we highlight the main limitations encountered when programming for a Tofino hardware switch.

1) Computational limitations: The number of operations that the P4 module in the switch can handle is limited in order to ensure a high-speed packet processing and minimize packet forwarding delays. Specifically, the Tofino 1 chipset permits a maximum number of 12 match action stages per pipeline.

Additionally, metadata in P4 are valid only for the current packet processing which requires the use of registers to save contextual information. In the hardware switch, we can have registers, but during one packet processing, only a single access to a given register is permitted. Consequently, updating a register can be done in only one step (reading the old value and writing the new one), but it is not possible to read the register, perform several other computations using the read value, and subsequently update the new computed value to the given register. This restricts and complicates the use of registers.

[3]https://www.edge-core.com/productsInfo.php?cls=1&cls2=5&cls3=181&id=335

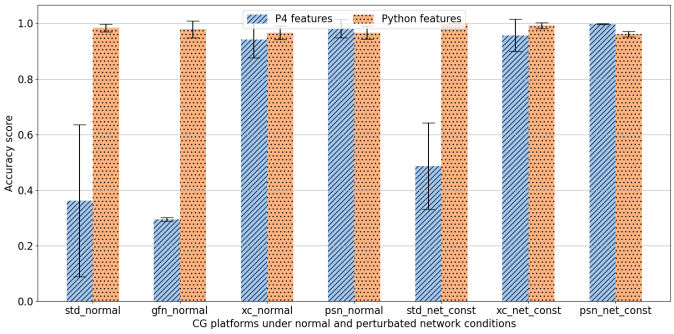

Fig. 4: Performance of the ML model with the standard deviation with P4-extracted features

The Tofino P4 provides a mathematical extern unit, called *MathUnit*, for complex functions like *SQR* and *SQRT*. However, a given register can have only a single *MathUnit* extern, and RegisterActions sharing a register can only use one MathUnit between them. Consequently, it is not possible to compute *SQR* and *SQRT* for the same value during the processing of one packet, thereby making the computation of the standard deviation impossible.

2) Computation limited to the current packet: The computation of the standard deviation feature poses another challenge. Since its computation is based on the mean value, we need to store in the memory all the packets within a time window and compute the mean and standard deviation at the end of the time window. However, the P4 Tofino hardware switch lacks a packet buffering/copying capability. To address this limitation, we explored the Welford's algorithm to compute the standard deviation online but the P4 Tofino hardware switch does not support variable multiplication and division. Consequently, we attempted to approximate the standard deviation through a simple computation. However, as depicted in Fig. 4, this approximation adversely affected the classification performance compared to the accurate computation of the standard deviation.

In light of these challenges, we decided to train the ML model without the two standard deviation features. Surprisingly, the simplified model exhibited strong performance and achieved results similar to the model with standard deviation, as illustrated in Fig.5. Removing the standard deviation features in the P4 extracting module then greatly simplifies the program and its P4 processing (only 6 stages are now necessary to process the packet). Moreover, the notable efficiency of this approach prompts us to retain this novel approach.

3) Size of Digest *message:* As mentioned in the beginning of this section, we use the *Digest* mechanism to send reports from the P4 module to the controller. However, the Tofino hardware restrics the size of the *Digest* message to 48 bytes. In our previous version with the standard deviation, we reached this maximum value when sending all the 12 features and had no space left to send the couple of IP addresses identifying the flow. Without the four standard deviation features (for the packet size and the IAT in both directions), we have now

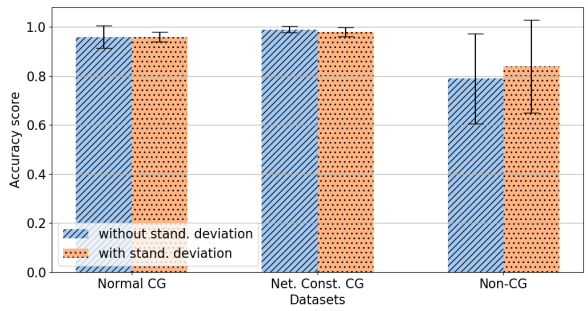

Fig. 5: Performance of the ML model with and without the standard deviation

enough space to include the IP addresses and port numbers.

4) Limited time-based processing: In the design of our ML model, we use time windows of 33 ms to split the session data in order to compute the features. This can be easily done with an application programming language such as Python or C++ using time handler, but not with P4. Indeed, a P4 program can trigger actions upon the reception of one packet, but not at a given time. Even if the Tofino offers the possibility to generate packets at a periodic interval, it is not suitable for our case. Indeed, the payload of the generated packet is set up by the configuration plane and even if modifying this payload is possible in the P4 program, we can not make it for all the sessions passing through the switch. Therefore, we opt for the *Digest* approach and manage this timing inside the P4 program by using one register to store the time when a first packet arrives and at each packet arrival, we check whether the time exceeds the required time window of 33 ms. If so, we send the report to the controller. It means that for regular and high bitrate sessions, for which we can expect very frequent packet arrivals, this way of computing the time window can be close to the 33 ms time, but for low-rate traffic, sporadic or bursty traffic, this approach is not appropriate because of possible long time intervals between packet arrivals. For NCG traffic, this limitation greatly degrades the performance of our solution since very few reports are sent over long periods, being non representative of the application traffic behavior. To circumvent it, we include a time handler in our controller module which generate *empty* reports and send them to the NFV compute node if the time interval between the reception of two reports is longer than two time windows of 33 ms. As such, the ML module can have reports, close to what it expects (i.e. a regular 33 ms basis). Despite being a major improvement, this solution is still not perfect since for example, if one packet arrives at $t+31$ ms and the next one at $t+46$ ms, the report sent by the P4 module will be related to a 46ms time window and not a 33ms one. Consequently, the ML model will not be aware that there was no traffic during the missing 13ms.

In conclusion, we have learned several lessons based on our experience with P4 implementation on a real hardware switch. Firstly, the P4 program cannot not perform many processing tasks for a single packet, cannot handle complex computational tasks and is not really suitable for programs with time based events. Therefore, careful consideration and optimization are required when developing a P4 code.

Secondly, a trade-off must be made between the high speed line-rate packet processing and the ease of programming, as well as the complexity of the computational tasks. Our proposed 2-level programmable architecture appears to be a promising approach to meet both factors, the developers splitting the processing adequately.

V. Evaluation

In this section, we present the findings of our experiments using the whole chain. The observed traffic passes through our P4 hardware switch, which extracts the features (excluding the four based on standard deviation) and then fed them to our unsupervised ML model trained without the standard deviation features.

Tab.I shows our evaluation results and compares them with the results obtained with an application-level computation in Python of all the features, including the standard deviations. The two classification scores are very close and remains excellent. In conclusion, having the feature extraction in a P4 module does not alter the performance of the ML model, although having simplified processing tasks and limitations as presented in the previous section.

This paper does not focus on the performance evaluation of the hardware switch because its line-rate processing performance is guaranteed in silico as long as the P4 code can be compiled and deployed. The hardware we use is a commercial switch[4] used in operational networks that can manage up to 6.4 Tbit/s (32x100G ports) enabling thousands of simultaneous sessions (being cloud gaming or not). Furthermore, Edgecore has recently launched a 12.8 Tb/s P4-programmable open switch, the DCS810, hosting an Intel Tofino 2 chipset.

Our evaluation proves that our P4 hardware module performs very well and can be considered as a candidate for possible ISP-level deployment.

VI. Conclusion

The main motivation behind this work was to classify CG traffic in a faster and more reliable way to enable latency sensitive network processing such as L4S. We first designed and evaluated an unsupervised model based on USAD which outperforms a supervised approach when dealing with new games, and above all, new CG platforms.

Then we proposed a hybrid architecture leveraging a P4 hardware implementation[5] on a Tofino chipset of the feature extraction module that must be executed in the data plane while keeping the ML model execution as VNFs in the control plane. This implementation was very challenging because of the very restrictive limitations engraved in a real P4 switch. Yet, we proposed several adaptations to circumvent the latter,

[4]https://www.edge-core.com/_upload/images/2022-051-DCS800_Wedge100BF-32X-R10-20220705.pdf

[5]https://github.com/mosaico-anr/P4_NFV_CG_Detector

TABLE I: Comparison of the P4-based solution performance with application-level solution

Traffic	Type	Performance with P4		Performance with application		Difference (%)	
		Accuracy	F1-score	Accuracy	F1-score	Accuracy	F1-score
Normal CG	**STD**	0.965	0.983	0.984	0.992	-1.90	-0.97
	GFN	0.990	0.995	0.979	0.989	1.16	0.58
	XC	0.903	0.942	0.966	0.981	-6.35	-3.85
	PSN	0.981	0.990	0.966	0.983	1.46	0.76
	Overall	**0.958**$_{\pm 0.054}$	**0.979**$_{\pm 0.03}$	**0.973**$_{\pm 0.021}$	**0.986**$_{\pm 0.011}$	**-1.44**	**-0.74**
CG with network constraints	**STD**	0.999	0.999	1.000	1.000	-0.01	-0.01
	XC	0.954	0.977	0.992	0.996	3.33	1.55
	PSN	0.995	0.998	0.961	0.983	-3.73	-1.91
	Overall	**0.983**$_{\pm 0.035}$	**0.993**$_{\pm 0.019}$	**0.984**$_{\pm 0.019}$	**0.992**$_{\pm 0.010}$	**-0.14**	**0.05**
Non CG	**VC**	0.938	0.971	0.867	0.941	7.18	3.00
	LV	0.980	0.990	0.978	0.989	0.17	0.10
	VS	0.993	0.996	0.991	0.996	0.14	0.01
	FB	0.988	0.994	0.989	0.995	-0.10	-0.04
	Overall	**0.959**$_{\pm 0.056}$	**0.983**$_{\pm 0.031}$	**0.918**$_{\pm 0.114}$	**0.970**$_{\pm 0.066}$	**4.13**	**1.30**
New games CG learned platforms	**GFN**	0.973	0.986	0.995	0.997	-2.15	-1.10
	XC	0.969	0.984	0.979	0.989	-0.97	-0.50
	PSN	0.980	0.990	0.999	1.000	-1.97	-1.00
	Overall	**0.974**$_{\pm 0.010}$	**0.987**$_{\pm 0.005}$	**0.991**$_{\pm 0.011}$	**0.996**$_{\pm 0.006}$	**-1.70**	**-0.84**
New CG platforms	**MoonLight**	0.999	0.999	1.00	1.00	-0.01	-0.01
	Steam	0.999	0.999	1.00	1.00	-0.01	-0.01
	Overall	**0.999**$_{\pm 0.00}$	**0.999**$_{\pm 0.00}$	**1.00**$_{\pm 0.00}$	**1.00**$_{\pm 0.00}$	**-0.01**	**-0.01**

that constitute in itself a valuable feedback for the community, and that allowed the execution on hardware while maintaining an excellent identification accuracy equivalent to the software implementation.

Our future work will integrate our classifier with a L4S architecture and evaluate the gain for CG platforms to respect their latency requirements under challenging network conditions. Also, we will evaluate our architecture with real traffic from an ISP to further assess the performance and classification results in real conditions.

ACKNOWLEDGMENTS

This work is partially funded by the French ANR MOSAICO project, No ANR-19-CE25-0012.

REFERENCES

[1] X. Marchal, P. Graff, J. R. Ky, T. Cholez, S. Tuffin, B. Mathieu, and O. Festor, "An analysis of cloud gaming platforms behaviour under synthetic network constraints and real cellular networks conditions," *Journal of Network and Systems Management*, vol. 31, 2023.

[2] P. Graff, X. Marchal, T. Cholez, B. Mathieu, and O. Festor, "Efficient Identification of Cloud Gaming Traffic at the Edge," in *NOMS 2023 - 36th IEEE/IFIP Network Operations and Management Symposium*, Miami, United States, May 2023, p. 10. [Online]. Available: https://hal.inria.fr/hal-04056607

[3] B. A. A. Nunes, M. Mendonca, X. N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Communications Surveys & Tutorials*, vol. 16, pp. 1617–1634, 2014.

[4] N. McKeown, T. E. Anderson, H. Balakrishnan, G. M. Parulkar, L. L. Peterson, J. Rexford, S. Shenker, and J. S. Turner, "Openflow: enabling innovation in campus networks," *Comput. Commun. Rev.*, vol. 38, pp. 69–74, 2008.

[5] P. W. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. E. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: programming protocol-independent packet processors," *Comput. Commun. Rev.*, vol. 44, pp. 87–95, 2013.

[6] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford, "Heavy-hitter detection entirely in the data plane," *Proceedings of the Symposium on SDN Research*, 2016.

[7] R. Harrison, Q. Cai, A. Gupta, and J. Rexford, "Network-wide heavy hitter detection with commodity switches," *Proceedings of the Symposium on SDN Research*, 2018.

[8] 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 18)*, 2018, pp. 1–6.

[9] K. A. Simpson, R. Cziva, and D. P. Pezaros, "Seiðr: Dataplane assisted flow classification using ml," in *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, 2020, pp. 1–6.

[10] Z. Xiong and N. Zilberman, "Do switches dream of machine learning?: Toward in-network classification," *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*, 2019.

[11] R. Kamath and K. M. Sivalingam, "Machine learning based flow classification in dcns using p4 switches," in *2021 International Conference on Computer Communications and Networks (ICCCN)*, 2021, pp. 1–10.

[12] G. Gombos, M. Mouw, S. Laki, C. Papagianni, and K. De Schepper, "Active queue management on the tofino programmable switch: The (dual)pi2 case," in *ICC 2022 - IEEE International Conference on Communications*, 2022, pp. 1685–1691.

[13] W. Zheng, C. Gou, L. Yan, and S. Mo, "Learning to classify: A flow-based relation network for encrypted traffic classification," *Proceedings of The Web Conference 2020*, 2020.

[14] J. Audibert, P. Michiardi, F. Guyard, S. Marti, and M. A. Zuluaga, "Usad: Unsupervised anomaly detection on multivariate time series," *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020.

[15] S. S. Johari, N. Shahriar, M. Tornatore, R. Boutaba, and A. Saleh, "Anomaly detection and localization in nfv systems: an unsupervised learning approach," *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–9, 2022.