Fast Learning Enabled by In-Network Drift Detection

Bruno Missi Xavier
Magnos Martinello
bmissix@gmail.com
Federal University of Espírito Santo/Brazil

Celio Trois
Federal University of Santa Maria/Brazil

Brenno Mello
Ricardo Rios
Federal University of Bahia/Brazil

ABSTRACT

The widespread adoption of Machine Learning (ML) is leading to an increase in processing demands. Dealing with the growing volume of data poses a significant challenge in providing accurate classification services using ML models. Offloading ML tasks to network switches presents an opportunity to tackle this challenge, offering high throughput and low latency. Nonetheless, network devices encounter limitations in resources, and programmable languages like P4 lack support for basic operations, necessary for the ML methods, including floating-point arithmetic and native repetition structures. In this paper, we investigate the use of drift detection ML models to enhance the accuracy of in-network traffic classification. The novelty lies in designing drift detection based on bitwise operations, which are well-suited for implementation within the data plane. As a proof-of-concept, we implement drift detection using the P4 language on BMv2 switches, validated with a dataset of over 2 million samples. Our results demonstrate a significant increase in classification accuracy with drift detection, while maintaining line-speed operation and quickly adapting to changes in traffic patterns.

ACM Reference Format:
Bruno Missi Xavier, Magnos Martinello, Celio Trois, Brenno Mello, and Ricardo Rios. 2024. Fast Learning Enabled by In-Network Drift Detection. In *The 8th Asia-Pacific Workshop on Networking (APNet 2024), August 3–4, 2024, Sydney, Australia.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3663408.3663427

ACM ISBN 979-8-4007-1758-1/24/08…$15.00
https://doi.org/10.1145/3663408.3663427

1 INTRODUCTION

Artificial Intelligence (AI) has experienced increasingly popularity leading to an increase in processing demands. Managing the increasing volume of data presents a considerable challenge when delivering accurate classification services using Machine Learning (ML) models. Currently, the learned models are typically deployed either on end-hosts or at the network control plane. However, recent progress in Programmable Data Plane (PDP) has enabled the introduction of in-network ML models to facilitate traffic analysis (such as attack detection) and enforce actions within high-speed switches [16, 23].

The network programmability approach shifts the burden of inference processes from servers or cloud to the programmable data plane. The primary advantage of performing model inference on the network data plane lies in its ability to provide *traffic analysis at line-speed* by leveraging data-driven models instead of predefined protocols [17, 19, 21]. Despite these previous successes, running ML within network switches has proven hard to tackle. While ML accelerators usually focus on matrix multiplication [7], network switches do not support such operations. Recent solutions have tried to address this limitation by modifying the data plane hardware or designing new hardware modules [22], but this is an experimental not off-the-shelf to be easily adopted.

Efforts to implement ML-based classification have been conducted on various platforms including network interface cards (NICs) [16], FPGA [15], and switch-ASIC [20]. However, network devices typically do not support ML functionality and face significant resource constraints making the development of ML models difficult. For instance, P4 lacks support for floating-point and repetition structures, as well as common operations required by ML algorithms such as division, polynomials, or logarithms, which are not natively supported in P4.

Another relevant aspect concerning ML applications in networks pertains to changes in the data distribution, commonly referred to as Concept Drift (CD) [3]. In this scenario, a CD occurs when the underlying problem's conditional probability $P(y|X)$ changes over time, thereby influencing the decision boundary and negatively impacting the learned ML model, i.e. the model becomes less accurate. These changes can be detected with the aid of drift detection algorithms [4], which trigger updates to the ML model.

In this paper, we explore the use of drift detection to signal changes and update the ML model, thus improving the overall accuracy of in-network traffic classification. The novelty lies in crafting drift detection mechanisms based on bitwise operations, deployable directly into the data plane. However, due to the constraints of data plane resources, some parameters cannot be calculated, making it necessary to estimate them. Our focus is on detecting CD whenever the traffic distribution changes, thus triggering the rebuilding of the In-Network classifier with the training phase executed at the control plane. This strategy allows the ML model to evolve over time, with training performed on-demand, thereby minimizing the need for costly continuous training.

To illustrate the benefits of employing a CD approach in classification systems, we contrast the traditional ML process, where the model is trained at the start of the experiment, with our proposed method focused on detecting changes in network traffic distribution in real-time and retraining the ML model each time the CD is detected. The In-Network Drift model, equipped with the capability for timely retraining, demonstrates resilience to fluctuating traffic distribution, swiftly adjusting its accuracy to acceptable levels.

2 DESIGN OF IN-NETWORK DRIFT

This section introduces a novel solution for CD within network, aiming for seamless integration with the data plane. It addresses architectural constraints and emphasizes a model-free, off-policy approach to adapt to CD properties efficiently. By aligning with the “action first update later” paradigm, the algorithm caters to the operational logic of programmable data planes, crucial for effective CD management. This architecture ensures compatibility while addressing CD challenges in network. The section first presents an overview of how the proposal was designed, showing its relations with the components of a programmable network. Finally, we explain the CD terminology and equations.

2.1 In-Network Components Workflow

Providing a comprehensive view of the system components, Figure 1 delineates a conceptual view of the proposed system, presenting its modules and their interconnections. Upon the arrival of new traffic to the *Data Plane*, the *ML Model* classifier promptly identifies the class to which the packet or flow belongs. Subsequently, the inferred prediction undergoes evaluation by the *Drift Detector*. If the component detects the need of updating the ML model, it signals the *Control Plane*, initiating a new round of training. This approach ensures that the system remains responsive to evolving network dynamics. Subsequently, leveraging the predicted traffic label, the switch executes appropriate policies, either forwarding or dropping packets.

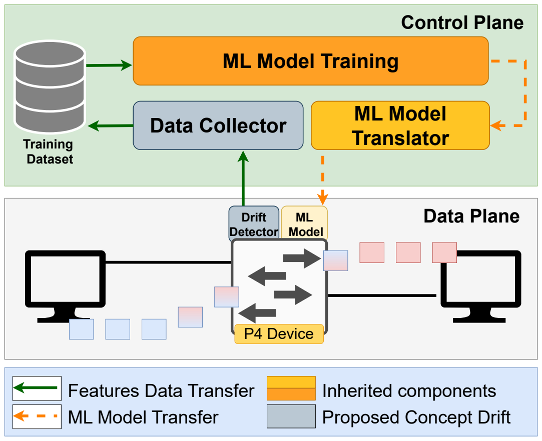


Figure 1: Structural components of the In-Network Concept Drift.

Built upon previously established framework [17], our *Control Plane* highlights two pivotal components that directly interface with the operational approach. Once the *Control Plane* receives an alert from the *Drift Detector*, new data starts to be collected and added to the *Training Dataset*. As soon as this dataset is ready, the model undergoes training conducted by the *ML Model Training* module and then translated by the *ML Model Translator*. Both phases adhere to the specifications outlined in [17], which involve transforming a Decision Tree model into a Match/Action pipeline. However, it is worth noting that our approach does not remains a close-loop, and it still concerns to the operator’s input to accurately identify traffic classes and determine the amount of data necessary for the *ML* training phase. These factors directly impact the accuracy of the resulting model.

2.2 Drift Detector

Concept Drift domain focuses on analyzing changes in the conditional distribution of the output of a target variable concerning the input. While it is possible to anticipate changes in certain real-world scenarios, CDs are typically deemed unexpected and unpredictable, as highlighted in [5].

Formally, the CD between two points at time t_0 and time t_1 can be defined as follows:

$$\exists X : p_{t0}(X, y) \neq p_{t1}(X, y), \tag{1}$$

in which p_{t0} is the coexisting distribution of the input variable X and the target variable y at time t_0. Similarly, p_{t1} is the coexisting distribution of the input variable X and the target variable y at time t_1, which must be different from p_{t0}.

Exponentially Weighted Moving Average (EWMA) [12] is a statistical method used for smoothing time series data to identify trends over time while reducing the impact of random fluctuations or noise. The weight assigned to each observation decreases exponentially as it moves further back in time. This means that recent observations have a greater influence on the calculated average than older ones. The calculation of EWMA is defined by the following equation.

$$ewma(y) = \sum_{i=t-1}^{0} \lambda y_t + (1-\lambda) ewma(y_i) \tag{2}$$

where, t is the current time, y_t is the current observation, and $ewma(y_i)$ is the EWMA calculated from the previous time steps. The λ is a constant in the range $[0, 1]$ representing the weight given to newer data compared to older data.

The observation here marks the accuracy of the model's inference of instance t followed by:

$$y_t = \begin{cases} 0 & \text{real label = predicted label} \\ 1 & \text{real label} \neq \text{predicted label} \end{cases} \tag{3}$$

Control limits play a crucial role in monitoring the process mean or variance over time and detecting shifts or trends in process performance. These control limits, typically derived from the standard deviation of the process data, help identify statistically significant deviations from the expected behavior of the process. The control limits of the EWMA are given by:

$$\sigma = \sqrt{\epsilon_t(1-\epsilon_t)\frac{\lambda}{2-\lambda}(1-(1-\lambda)^{2t})} \tag{4}$$

where ϵ is the percentage accuracy error found.

The Average Run Lenght (ARL_0) represents the average number of observations needed to signal an out-of-control condition, indicating a detected process shift. ARL_0 serves as a performance metric, assessing the EWMA's effectiveness in detecting changes in process behavior while minimizing false alarms; however, computing it is computationally expensive. Notably, researchers have developed polynomial approximations for various ARL_0 values to aid in chart interpretation and analysis [12]. As an illustration, considering an ARL_0 of 400, means that it is desired to maintain a rate of 1 false positive per 400 data points. $ARL_0(400)$ is defined by:

$$ARL_0(400) = 3.97 - 6.56y_t + 48.73{y_t}^3 - 330.13{y_t}^5 + 848.18{y_t}^7 \tag{5}$$

Algorithm 1 showcases the CD pseudo-code, according to [12]. Observations (packets or flows) in the data stream are processed sequentially by the classifier (line 3), with each observation denoted as X_t. If the predicted class label clX_t is correct, y_t is set to 0; otherwise, it is set to 1 (line 4). In line 6 the estimate of the standard deviation, σ, is then updated based on the value of ϵ_t. Subsequently, a polynomial from Equation 5 is utilized to determine the control limit, L_t, corresponding to the desired ARL_0 in line 7. The EWMA estimator, Z_t, is updated accordingly (line 8). If Z_t exceeds the sum of the error term, ϵ_t, and L_t times σ (line 9), it is inferred that CD has occurred.

3 IMPLEMENTATION DECISIONS

P4 language presents significant challenges in deploying ML algorithms into the data plane due to the stringent constraints on floating-point operations, such as divisions, and the absence of loop structures. However, leveraging highly

Algorithm 1 Concept Drift Detection algorithm.

```
Require: Choose a desired value for $\lambda$ and the $ARL_0$;
Ensure: Initialize the classifier;
 1: $Z_0 = 0$; $\epsilon_0 = 0$; $sum\_err = 0$;
 2: for each observation $X_t$ do
 3:     $clX_t$ = Classify($X_t$);
 4:     $y_t = 0$ if $clX_t$ was correctly classified else 1;
 5:     $sum\_err += y_t$; $\epsilon_t = sum\_err/t$;
 6:     $\sigma = \sqrt{\epsilon_t(1-\epsilon_t)\frac{\lambda}{2-\lambda}(1-(1-\lambda)^{2t})}$;
 7:     Compute $L_t$ using Equation 5;
 8:     $Z_t = \lambda y_t + (1-\lambda)Z_{t-1}$; {calculates the EWMA}
 9:     if $Z_t > \epsilon_t + \sigma \cdot L_t$ then
10:         Flag for concept drift;
11:         $Z_0 = 0$; $\epsilon_0 = 0$; $sum\_err = 0$;
12:     end if
13: end for
```

```
action ewma(bit<32> X) {
  bit<32> ewma_ = X - meta.ewma;
  meta.ewma=ewma_+(ewma_>>12)+(ewma_>>15)+
      (ewma_>>16)+(ewma_>>17)+(ewma_>>19);
}
```

Listing 1: P4 code of EWMA building block as described in Equation 2, with λ defined in Equation 6.

recommended bitwise operations offers promising avenues to overcome these limitations. We employ binary long division and left / right shifts to compute the necessary estimators (i.e., error rate, and EWMA).

Our first decision is to establish fixed values for the parameters λ and ARL_0. According to the results of [12], $ARL_0 = 400$ gives competitive performance for this type of gradual drift. In our numerical parameterization, we choose a value of $\lambda = 0.0003$ for EWMA, emphasizing the importance of early detection, particularly crucial to identify malicious activities.

A common optimization strategy when multiplying a variable by a constant, involves converting the constant into a power of 2 [8] whenever possible, simplifying the multiplication to a left or right shift operation. In our case, the use of $\lambda = 0.0003$ can be broken down into the following series:

$$\lambda = 1/2^{12} + 1/2^{15} + 1/2^{16} + 1/2^{17} + 1/2^{19} \approx 0.0002995 \tag{6}$$

so that it can be used to calculate the EWMA. Listing 1 showcases the P4 code building block that applies EWMA setting this value to λ.

In order to tailor the CD algorithm to P4, since we are shifting right up to 19 bits to calculate the λ, we apply a left shift operation of 25 bits on the error amount sum_err and the current observation y_t (Listing 2, lines 8 and 10); this makes the significant bits of these variables to be maintained when performing the right shifting during the EWMA calculation.

```
1  apply {
2    read_registers(); //from the registers
3    classify(); //Classify according to [19]
4    bit<32> pr = (meta.class == meta.pred) ? 0
         : 1;
5    meta.m_sum = meta.m_sum + pr;
6    meta.m_p = 0;
7    //Stores the division into meta.m_p var
8    divide((meta.m_sum << 25), meta.m_n);
9    meta.m_n = meta.m_n + 1;
10   ewma(pr << 25);
11   write_registers(); //Into the registers
12   err_P4 =  meta.m_p + meta.m_p >> 2;
13   if meta.ewma > err_P4 { //Flag for CD
14       hdr.int_headers[0].drift = 1;
15       reset_registers();
16   }
17 }
```

Listing 2: P4 code showcasing the necessary building blocks to perform the drift detection.

An estimation strategy involves defining values for the parameters $\sigma \cdot L_t$. Through an analysis of the values generated by these parameters, we observe that, in 97.4% of the instances, there is a relation of $\epsilon + \sigma \cdot L_t \approx \epsilon + \epsilon * 0.2668$. In our implementation, we simplify the division by right shifting 2 bits ($1/2^2 = 0.25$). This adaptation is presented in Listing 2, line 12, along all building blocks necessary for developing the In-Network CD to the P4 language. While we acknowledge that this is a data-specific approximation, it represents a feasible step towards generalizing to any dataset. It is important to highlight that every packet crossing the device is processed by this specific P4 code.

4 EXPERIMENTAL RESULTS

In this section, we report the experiments conducted for assessing our In-Network Drift Detection approach[1]. First, we detail the dataset used in the evaluation process, next we clarify the experimental setup, including the classification granularity, traffic distribution over time, and emulation environment. Lastly, we present the detected drifts in traffic distribution, discussing the accuracy of our approach.

4.1 Dataset

For our experiments, we used the dataset provided by [14], which comprises PCAP files that capture network traffic over a 5-day period (Monday to Friday). The flows in the dataset are categorized as either benign or representing specific types of attacks. The authors demonstrate the effectiveness of ML in accurately distinguishing between different flow types using well-established algorithms and carefully designed feature extraction and selection techniques. Is not the scope of this paper to delve into the specifics of features, their generation, or their utilization for classification purposes.

[1]Our source code is publicly available at github.com/nerds-ufes/

Table 1: Dataset Summary

Class	N. of Samples
Benign	1.839.070
DoS Hulk (DH)	498.362
DoS GoldenEye (GE)	66.795
DoS SlowHttpTest (SH)	32.510
DoS SlowLoris (SL)	37.236
PortScan (PS)	174.312
BruteForce (BF)	19.755
SQL Injection (SI)	67
WA XSS (WX)	6.361
Total	**2.674.468**

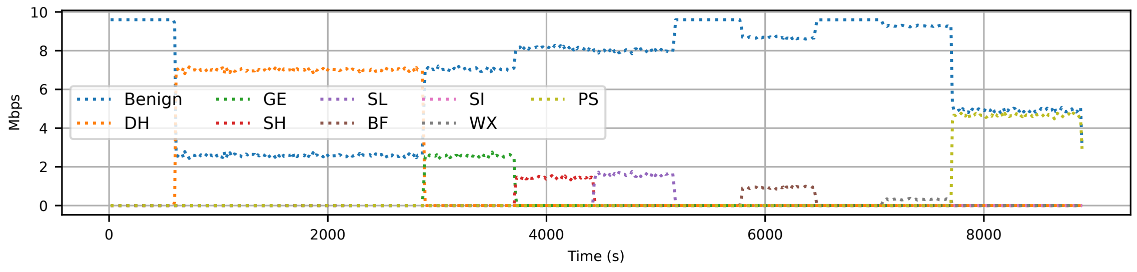
From the available dataset, we opted to focus on the PCAP files from two specific days (Wednesday and Thursday), which contain flows related to seven distinct types or classes: one labeled as "benign" (BE) and eight representing various types of attacks, namely "DoS Hulk" (HK), "DoS GoldenEye" (GE), "DoS SlowHttpTest" (SH), "DoS SlowLoris" (SL), "PortScan" (PS), "Brute-force" (BF), "SQL Injection" (SI), and "Web Attack Cross Site Scripting" (WS). For a concise overview of the dataset utilized in our study, please refer to Table 1.

4.2 Experimental Setup

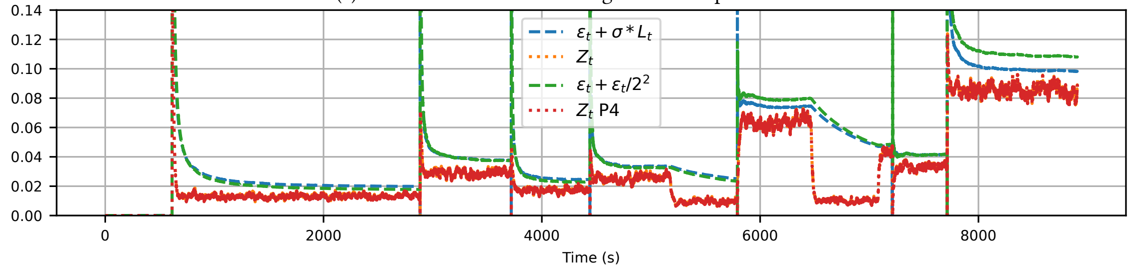
Experiments were carried out using an implementation of Algorithm 1 in Python and our approach deployed in a BMv2 emulator [11], with the objective of evaluating the quality of the CD and observing the behavior of the P4 code in a controlled setting. In order to conduct our CD experiments, we directed the benign traffic available in the dataset through the P4 switch, periodically introducing various types of attacks among benign traffic flow.

Figure 2(a) visually depicts the evolution of the traffic distribution over time; initially, pure benign traffic dominates the network throughput. In the second period, the *DoS Hulk* attack is injected amidst the benign traffic. Subsequently, the burst of *DoS Hulk* finishes, giving way to the emergence of *DoS GoldenEye* in the third period. Following the conclusion of *DoS GoldenEye*, *DoS SlowHttpTest* initiates its attack within benign traffic in the fourth period. The fifth period sees the onset of *DoS SlowLoris* traffic. The sixth period reverts to pure benign traffic, mirroring the initial stage. In the seventh period, a burst of *BruteForce* attack occurs. The eighth period witnesses the introduction of *SQL Injection*, followed by *WA XSS* in the ninth period. Lastly, the tenth period is characterized by the attack *PortScan*, which floods the network with malicious packets.

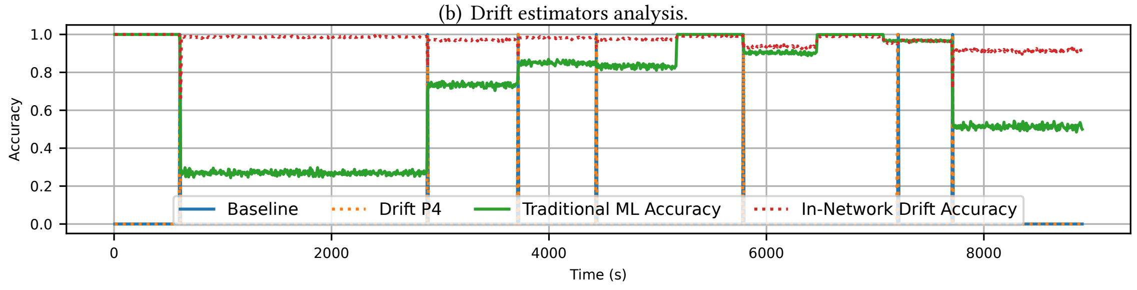
In network traffic analysis, there exists a trade-off between per-packet and per-flow models [16]. Per-packet granularity entails a leaner ML model, as it extracts features solely from packet headers. Additionally, this approach aligns well with the instantaneous transmission of packets. Conversely,



(a) Distribution of classes along the time experiment.



(b) Drift estimators analysis.



(c) Classification Accuracy with drift detection

Figure 2: In-Network Drift Detection experimental results.

per-flow features can be derived by aggregating data from multiple packets. Leveraging these comprehensive features often results in superior model training compared to using individual packets. Hence, a per-packet model has the potential to be more efficient, even though it does not offer to the model aggregated measurements and the possibility of learning from temporal correlation. So, in the scope of this work we employed per-packet for classifying the traffic.

4.3 Drift Detection

In the initial phase, a ML Decision Tree model, trained only with benign data is deployed in the P4 switch. Following the deployment of the ML model into the P4 switch, we initiate traffic transmission and proceed to gather results. Upon the detection of a change in traffic distribution (Figure 2(a)) by the *Drift Detector*, a new training phase initiates, accumulating data for further ML model training.

To visualize the drift detection, Figure 2(b) illustrates a comparison between values calculated in Python, represented by Z_t in dotted orange and $\epsilon_t + \sigma \cdot L_t$ in dashed blue, and our P4 implementation, shown as $Z_t P4$ in dotted red and $\epsilon_t + \epsilon_t/2^2$ in dashed green. Notably, to place on the same scale, the P4 values were divided by 2^{25}. It's worth mentioning that $Z_t P4$ closely aligns with the values of Z_t. Upon comparing the values, an average difference of 0.000279 is observed, attributed to the approximation provided by Equation 6.

Comparing the drift triggers between the Python and P4 versions reveals distinct relations: $Z_t > \epsilon_t + \sigma \cdot L_t$ for Python (see Algorithm 1, line 9) and $Z_t P4 > \epsilon_t + \epsilon_t/2^2$ for P4 (Listing 2, line 13). Notably, our approach estimates smaller values than Python when the error rate is lower (seen around 2000s in Figure 2(b)). Conversely, as the error rate increases (observed after 8000s), the estimated values become higher. This behavior is expected since our implementation leverages ϵ_t to estimate $\sigma \cdot L_t$, resulting in an estimated $\epsilon_t + \epsilon_t/2^2$, proportionally 1.25 greater than the error rate ϵ_t.

Figure 2(c) displays the CD detection implemented in Python, defined as *Baseline* (vertical solid blue line); in contrast, *Drift P4* shows our In-Network CD (vertical dotted orange), highlighting the challenges posed by the P4 language. It is evident that the suggested drifts between both approaches closely align, although not exactly matching. For instance, the sixth drift identified by *Baseline* is not mirrored by *Drift P4*, which flags the drift 1,247 instances ahead.

Table 2: Summary of classification results

Class	Precision	Recall	F1-Score
Benign	0.97	0.98	0.98
DoS Hulk	0.98	1.00	0.99
DoS GoldenEye	0.90	1.00	0.95
DoS SlowHttpTest	0.94	0.94	0.94
DoS SlowLoris	0.91	0.89	0.90
PortScan	0.96	0.85	0.91
BruteForce	0.51	0.43	0.47
SQL Injection	0.00	0.00	0.00
XSS	0.00	0.00	0.00

This discrepancy stems from certain approximations in the estimators, denoted by σ and L_t (discussed in Section 3). Consequently, at this point, the reduction in accuracy was significant enough to be recognized as a delayed drift.

4.4 Accuracy Improvement

To elucidate the advantages of employing a CD approach in classification systems, we juxtapose the traditional ML process with the proposed In-Network CD. Here, the traditional model approach is trained at the experiment's onset. Conversely, In-Network CD is geared toward real-time detection of changes in network traffic distribution and on-demand ML model retraining. Figure 2(c) presents a comparison of the accuracy achieved by both approaches: the traditional ML process depicted in green and the In-Network CD method shown in red.

As depicted in Figure 2(c), the traditional ML process struggles to classify new traffic patterns due to its static training nature, which relies solely on data collected during the initial phase (as illustrated in Figure 2(a)). Conversely, the In-Network Drift model, equipped with the capability for timely retraining, demonstrates resilience to fluctuating traffic distribution, swiftly adjusting its accuracy to acceptable levels. Notably, the model encounters challenges in classifying instances of *SQL Injection* and *WA XSS*, primarily attributed to their infrequent occurrence in the dataset. Further insights into the accuracy of the In-Network CD model post-experiment are detailed in Table 2. Notably, misclassification rates are observed for *BruteForce* and *PortScan*, largely due to traffic features that are better aligned with the characteristics of a per-flow model, such as BytePackets/s and BytePacket Len Std, as discussed in [14].

5 RELATED WORK

Recent research have underscored the remarkable efficacy of the CD in discerning crucial shifts within network traffic distribution [10, 18, 24]. Drift detection, a technique derived from this concept, has found widespread application in the realm of security, particularly in swiftly identifying potential attacks within a time windows [1, 9].

INSOMNIA [2] and Seth et al. in [13] integrate CD into the IDS scenario, using the CICIDS2017 dataset [14] of traffic traces categorized into various attack types, represented by high-level statistical features. INSOMNIA [2] employs active learning to reduce model update latency and label estimation to decrease labeling overhead. In [13], the authors introduce an Adaptive-Random Forest classifier coupled with the ADWIN change detector approach for online intrusion detection, using stream-oriented learning to effectively adapt to CD in real-world environments.

Jain et al. [6] propose a hybrid CD detection technique for anomaly detection, merging K-Means clustering with SVM-based classification. They develop two drift detection methods: Error Rate Based CD and Data Distribution Based CD. Detection accuracy, KL-Divergence, and Kappa Statistics gauge the severity of concept drift, showing high classification accuracy and precision rates across three datasets.

Our approach introduces an In-Network perspective using PDP to enable streaming learning methodologies. We assess drift concerning model accuracy, providing insights into its temporal evolution. To our understanding, our contribution marks an initial stride towards a practical integration of streaming learning techniques into the P4 language pipeline.

6 CONCLUSIONS AND FUTURE WORK

In conclusion, this paper has delved into the utilization of drift detection mechanisms to prompt updates the ML model, thereby enhancing the overall accuracy of in-network traffic classification. The novelty of our approach lies in the development of drift detection mechanisms based on bitwise operations, making them highly compatible for implementation within the data plane of network switches.

Acknowledging the limitations imposed by the resource constraints of the data plane, we understand that not all model mappings will be viable. Consequently, our attention has been directed towards reconstructing a decision tree, with the training phase conducted at the control plane whenever a drift detector is activated. This approach facilitates a split deployment strategy, where a streamlined model functions on the switch, while a more sophisticated model operates on the backend (control plane).

In comparison to the traditional ML process, which involves training the model previously, our proposed method demonstrates significant advantages. By detecting changes in network traffic distribution in real-time and retraining the ML model accordingly, the In-Network CD model exhibits resilience to fluctuating traffic patterns, swiftly adapting its accuracy to maintain acceptable levels. Future work on this topic involve generalizing the estimator σ, thereby reducing the dependency on data-specific approximations.

ACKNOWLEDGEMENT

Financial support from Brazilian agencies: CNPq, CAPES, FAPESP/MCTI/CGI.br (PORVIR-5G 20/05182-3, FAPES (1026/2022). CNPq fellows Dr. Martinello 312058/2023-3.

REFERENCES

[1] Emad Alsuwat, Suhare Solaiman, and Hatim Alsuwat. 2023. Concept Drift Analysis and Malware Attack Detection System Using Secure Adaptive Windowing. *Computers, Materials & Continua* 75, 2 (2023).

[2] Giuseppina Andresini et al. 2021. Insomnia: Towards concept-drift robustness in network intrusion detection. In *Proceedings of the 14th ACM workshop on artificial intelligence and security*. 111–122.

[3] Fabrício Ceschin, Marcus Botacin, Albert Bifet, Bernhard Pfahringer, Luiz S Oliveira, Heitor Murilo Gomes, and André Grégio. 2020. Machine learning (in) security: A stream of problems. *Digital Threats: Research and Practice* (2020).

[4] Joao Gama et al. 2004. Learning with drift detection. In *Advances in Artificial Intelligence–SBIA 2004: 17th Brazilian Symposium on Artificial Intelligence, Sao Luis, Maranhao, Brazil, September 29-Ocotber 1, 2004. Proceedings 17*. Springer, 286–295.

[5] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A survey on concept drift adaptation. *ACM computing surveys* (2014).

[6] Meenal Jain, Gagandeep Kaur, and Vikas Saxena. 2022. A K-Means clustering and SVM based hybrid concept drift detection technique for network anomaly detection. *Expert Systems with Applications* 193 (2022), 116510.

[7] Norman P Jouppi et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*. 1–12.

[8] Bojan Jovanovic, Ruzica Jevtic, and Carlos Carreras. 2013. Binary division power models for high-level power estimation of FPGA-based DSP circuits. *IEEE Transactions on Industrial Informatics* 10, 1 (2013), 393–398.

[9] Aditya Kuppa and Nhien-An Le-Khac. 2022. Learn to adapt: Robust drift detection in security domain. *Computers and Electrical Engineering* 102 (2022), 108239.

[10] Weike Liu, Cheng Zhu, Zhaoyun Ding, Hang Zhang, and Qingbao Liu. 2023. Multiclass imbalanced and concept drift network traffic classification framework based on online active learning. *Engineering Applications of Artificial Intelligence* 117 (2023), 105607.

[11] P4 Language Consortium. [n. d.]. P4-bmv2 Website. ([n. d.]). https://github.com/p4lang/

[12] Gordon J Ross, Niall M Adams, Dimitris K Tasoulis, and David J Hand. 2012. Exponentially weighted moving average charts for detecting concept drift. *Pattern recognition letters* 33, 2 (2012), 191–198.

[13] S Seth, G Singh, and K Chahal. 2021. Drift-based approach for evolving data stream classification in Intrusion detection system. In *Proceedings of the Workshop on Computer Networks & Communications, Goa, India*. 23–30.

[14] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. 2018. Toward generating a new intrusion detection dataset and intrusion traffic characterization.. In *ICISSP*. 108–116.

[15] Giuseppe Siracusano et al. 2022. Re-architecting traffic analysis with neural network interface cards. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 513–533.

[16] Bruno Missi Xavier, Rafael Silva Guimarães, Giovanni Comarela, and Magnos Martinello. 2021. Programmable Switches for in-Networking Classification. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*.

[17] Bruno Missi Xavier, Rafael Silva Guimarães, Giovanni Comarela, and Magnos Martinello. 2022. MAP4: A Pragmatic Framework for In-Network Machine Learning Traffic Classification. *IEEE Transactions on Network and Service Management* (2022). https://doi.org/10.1109/TNSM.2022.3212913

[18] Qiuyan Xiang, Lingling Zi, Xin Cong, and Yan Wang. 2023. Concept Drift Adaptation Methods under the Deep Learning Framework: A Literature Review. *Applied Sciences* 13, 11 (2023), 6515.

[19] Changgang Zheng, Benjamin Rienecker, and Noa Zilberman. 2023. QCMP: Load Balancing via In-Network Reinforcement Learning *(FIRA '23)*. Association for Computing Machinery, New York, NY, USA.

[20] C. Zheng, Z. Xiong, T. T. Bui, S. Kaupmees, R. Bensoussane, A. Bernabeu, S. Vargaftik, Y. Ben-Itzhak, and N. Zilberman. [n. d.]. IIsy: Hybrid In-Network Classification Using Programmable Switches. *IEEE/ACM Transactions on Networking* ([n. d.]). https://doi.org/10.1109/TNET.2024.3364757

[21] Changgang Zheng and Noa Zilberman. 2021. Planter: seeding trees within switches *(SIGCOMM '21)*. Association for Computing Machinery, New York, NY, USA.

[22] Zhizhen Zhong, Weiyang Wang, Manya Ghobadi, Alexander Sludds, Ryan Hamerly, Liane Bernstein, and Dirk Englund. 2021. IOI: In-network Optical Inference *(OptSys '21)*. Association for Computing Machinery, New York, NY, USA.

[23] Guangmeng Zhou, Zhuotao Liu, Chuanpu Fu, Qi Li, and Ke Xu. 2023. An Efficient Design of Intelligent Network Data Plane. In *32nd USENIX Security Symposium (USENIX Security 23)*. USENIX Association, Anaheim, CA, 6203–6220.

[24] Ming Zhou, Jie Lu, Yiliao Song, and Guangquan Zhang. 2023. Multi-Stream Concept Drift Self-Adaptation Using Graph Neural Network. *IEEE Transactions on Knowledge and Data Engineering* (2023).