ADAM & RAL: Adaptive Memory Learning and Reinforcement Active Learning for Network Monitoring

Sarah Wassermann*, Thibaut Cuvelier†, Pavol Mulinka‡, Pedro Casas*
*AIT Austrian Institute of Technology, †CentraleSupélec, ‡CTU Czech Technical University in Prague

***Abstract*—Network-traffic data commonly arrives in the form of fast data streams; online network-monitoring systems continuously analyze these kinds of streams, sequentially collecting measurements over time. Continuous and dynamic learning is an effective learning strategy when operating in these fast and dynamic environments, where concept drifts constantly occur. In this paper, we propose different approaches for stream-based machine learning, able to analyze network-traffic streams on the fly, using supervised learning techniques. We address two major challenges associated to stream-based machine learning and online network monitoring: (i) how to dynamically learn from and adapt to non-stationary data and patterns changing over time, and (ii) how to deal with the limited availability of ground truth or labeled data to continuously tune a supervised learning model. We introduce ADAM & RAL, two stream-based machine-learning approaches to tackle these challenges. ADAM implements multiple stream-based machine-learning models and relies on an adaptive memory strategy to dynamically adapt the size of the system's learning memory to the most recent data distribution, triggering new learning steps when concept drifts are detected. RAL implements a stream-based active-learning strategy to reduce the amount of labeled data needed for stream-based learning, dynamically deciding on the most informative samples to integrate into the continuous learning scheme. Using a reinforcement learning loop, RAL improves prediction performance by additionally learning from the goodness of its previous sample-selection decisions. We focus on a particularly challenging problem in network monitoring: continuously tuning detection models able to recognize network attacks over time. By continuously learning from and detecting concept drifts within real network measurements, we show that ADAM & RAL can continuously achieve high detection accuracy and limit the amount of training data needed to detect attacks over dynamic network data streams.**

***Index Terms*—Stream-based Machine Learning; Active Learning; Reinforcement Learning; ADWIN; Network Attacks; MAWILab.**

I. Introduction

Network-traffic monitoring and analysis is paramount to understand the functioning of complex large-scale networks, especially to get a broader and clearer visibility of unexpected events. One of the major challenges faced by online network-monitoring applications is the processing and analysis of large amounts of heterogeneous and fast incoming network-monitoring data. This sort of data usually comes in the form of high-speed streams, which need to be rapidly and continuously processed. In this context, detecting and adapting to strong variations in the underlying statistical properties of the modeled data makes data-stream analysis a very difficult task.

The application of machine-learning models to network-security and anomaly-detection problems has largely increased in the last decade; however, the general approach in the literature still considers the analysis as an offline learning problem, where models are trained once and then applied to the incoming measurements. This approach is very restrictive when dealing with highly dynamic environments, where concept drifts – i.e. changes in the underlying properties of the prediction target – occur often and previous knowledge rapidly becomes obsolete. An additional challenge of learning in in-the-wild networking scenarios is the lack or limited availability of ground truth or labeled data for training purposes. Labeling new incoming data is often an expensive and cumbersome process – especially when done manually and in an online fashion –, and not all data samples are equally valuable.

In this paper, we investigate stream-based approaches to machine-learning-based network security, using different algorithms for the analysis of continuously evolving data. Stream machine-learning analysis consists of processing one data instance at a time, inspecting it only once, and as such, using a limited amount of memory; stream approaches work in a limited amount of time, and have the advantage of being able to perform a prediction at any point in time during the stream.

We introduce novel stream-based, continuous learning strategies to deal with the aforementioned challenges; we conceive, describe, and evaluate ADAM & RAL, two stream-based machine-learning approaches to deal with (i) concept drifts in the stream of network measurements and (ii) limited availability of labeled, ground-truth measurements. ADAM relies on simple data-distribution change-detection algorithms to dynamically adapt the learning memory of different stream-based machine-learning models to the most recent data distribution, triggering new learning steps when concept drifts are detected. RAL consists of a stream-based active-learning strategy to reduce the amount of labeled data needed for learning, dynamically deciding on the most informative measurements to integrate into the continuous learning scheme. Active learning aims at labeling only the most informative samples to reduce the overall training cost. There is an assorted list of data-querying strategies and algorithms to decide which data samples should be labeled [28]; among them, the most popular strategy is based on uncertainty sampling, which

978-3-903176-24-9

uses the model-prediction uncertainty for the corresponding sample to decide whether to query its label or not. The higher the uncertainty of the model for a given sample, the more interesting the label of this sample becomes for adapting the model. RAL improves model training and prediction performance by additionally learning from the goodness of its previous sample-selection decisions, using a reinforcement-learning scheme. We make RAL freely available on GitHub as a Python package[1].

We evaluate the performance of the proposed approaches on the detection of different types of network attacks and anomalies, using real network measurements collected at the WIDE backbone network, relying on the well-known MAWILab dataset for attack labeling [2]. Results not only show that particular stream-based machine-learning models are able to keep up with important concept drifts in the underlying network data streams while keeping high detection accuracy, but also that it is possible to drastically reduce the amount of labeled data with stream-based active-learning approaches by relying on reinforcement-learning principles.

The remainder of this paper is structured as follows. Section II presents an overview on the related work. Sections III and IV introduce the proposed ADAM and RAL approaches. Section V presents evaluation results on the continuous detection of network attacks and anomalies on real network measurements, using both ADAM and RAL with different machine-learning models. Finally, Section VI concludes the paper.

II. State of the Art

The application of machine learning to networking problems has been largely explored in the literature [1]. There are a couple of extensive surveys on any-domain anomaly-detection techniques [4] as well as network-oriented anomaly detection [5], [6], including machine-learning-based approaches. We refer the interested reader to [1] for a detailed survey on the different machine-learning techniques commonly applied to network-traffic analysis. There are multiple recent papers on the application of machine-learning models to network-security and anomaly-detection problems [3], [7]–[9]. In [3], we analyze and benchmark big-data-analytics frameworks for large-scale network-traffic monitoring and analysis. In [7], we compare the performance of standard, offline machine-learning models for network security in fixed-line networks, further studying more complex and robust models based on ensemble-machine-learning techniques. Wireless-network monitoring using similar techniques is studied in [9]. In [8], we introduce GML learning, a generic machine-learning model for network-measurements analysis, which achieves high accuracy for many different network-analysis problems. However, all these approaches consider the offline analysis of network measurements, in batch mode.

The specific application of stream-based machine-learning approaches to network security and anomaly detection is by far more limited; a relevant and representative example linked to current research is presented in [10], where the authors evaluate stream-based traffic-classification approaches based on Hoeffding adaptive trees [20], using MAWILab data and the MOA machine-learning toolkit, as we do in this work.

Naturally, the data-stream machine-learning domain has a long-standing tradition and many interesting references are worth mentioning when considering the application and evaluation of stream-based machine-learning models; these cover general problems related to the learning properties for stream-based algorithms [11], [12], the mining and evaluation processes when dealing with massive datasets [13], the identification of model-evaluation issues [14], as well as propositions of general frameworks for data streaming [15]. Of particular relevance for stream-based machine-learning-model evaluation are the problems of *class imbalance* and *concept drift*, which are extensively addressed in [16].

When it comes to active learning, there is a vast literature in the field. For example, [33], [34] present three simple approaches for active learning. Their proposed Randomized Variable Uncertainty approach tackles the problem of stream-based active learning, using the model's prediction uncertainty to decide whether to query and trying to detect concept drifts by randomizing the certainty threshold used for labeling decisions. [32] develops an active-learning algorithm with two different classifiers: one "reactive" and one "stable". The stable classifier is trained on all available labeled instances, while the reactive one is trained based on a window of recent instances. [26] presents an active-learning technique based on clustering and prediction uncertainty. [27] conceives an approach relying on a modification of the Naïve Bayes classifier to update the different learners through the queried samples. In particular, they use one-versus-one classifiers to tackle multi-class problems and update the weights of the different classifiers by comparing their predictions to the ground truth. Their technique behaves similarly to RAL. However, the major difference is that [27] uses information about the classifiers' prediction certainty (without considering the corresponding weights) to adapt the minimum threshold for querying the oracle, while we rely on the usefulness of the decisions taken by RAL to tune the system according to the data stream.

Finally, ideas from reinforcement learning have already infused into the active-learning domain, but mostly into pool-based approaches. In [24], [25], authors rely on the multi-armed bandit paradigm. [25] develops ALBL, which uses a modified version of EXP4 [23], a weight-updating rule, in order to attribute adaptive weights to different learners based on rewards; the learner to use is then determined through these weights and uses its uncertainty measure to select the samples in the pool to hand to the oracle. The approach described in [24] is similar to the one in [25], except for the reward-computation scheme. The algorithm presented in [29], [30] relies on the same principles as the approach we are proposing, but tackles a different problem: Song's goal is to introduce an active-learning component into a contextual-bandit problem, while we are aiming at solving an active-learning problem by

[1]https://github.com/SAWassermann/RAL

using contextual bandits.

III. ADAM – Stream Learning with Adaptive Memory

We start by introducing ADAM, relying on an ADAptive Memory strategy. Given that we are dealing with continuous data analysis, the approach must be able to identify and adjust to the variation of the statistical properties of the analyzed data, detecting sudden statistical changes or concept drifts. To do so, ADAM relies on ADWIN, a dynamically adjusting window-based approach introduced in [18], which maintains a window of variable size containing training samples. The algorithm automatically grows the window when no change is apparent, and shrinks it when the statistical properties of the stream changes. ADWIN automatically adjusts its window size to the optimum balance point between reaction time and small variance; using the ADWIN adaptation strategy, ADAM implements four stream-based machine-learning algorithms popular in the literature, including incremental k-NN, Hoeffding adaptive trees (HAT), adaptive random forests (ARF) [21], and SVM through stochastic gradient descent (SGD).

A. Adaptation Strategy

One of the key features of a stream-based machine-learning model is that of continuously adapting to the changes of the underlying statistics describing the current data under analysis, by periodically re-training or re-calibrating. To deal with evolving data, one needs to define strategies for: firstly, detecting when changes occur; secondly, deciding which data to use for a subsequent model re-calibration (or, more generally, keeping updated sufficient statistics); and finally, re-training or re-calibrating the model when a significant change has been detected. Most strategies use variations of a simple sliding window approach, where a window containing the most recent measurements is kept and constantly updated, removing older measurements based on some specific criteria.

The ADWIN algorithm keeps a sliding window W with the most recently observed measurements x_n. At each time n, instance x_n is generated according to an unknown probability distribution D_n. Let m be the length of W, $\hat{\mu}_W$ the (computed) average of the measurements in W. The idea of ADWIN is straightforward: whenever two *large enough* sub-windows of W exhibit *distinct enough* averages, we conclude that the corresponding expected values are different, and the older portion of the window is dropped. Algorithm 1 briefly describes the algorithm, where $\hat{\mu}_{W_0}$ and $\hat{\mu}_{W_1}$ are the averages of the instances in W_0 and W_1, respectively. Note that it is not needed to define W_0 and W_1 sizes, as the values of those are decided by the algorithm itself. ADWIN is therefore a simple statistical test for different distributions in W_0 and W_1, which checks whether the observed average in both sub-windows differs by more than the threshold ϵ. This threshold is defined based on the global error of the statistical test and the lengths m_0 and m_1 of the corresponding sub-windows.

Algorithm 1 ADWIN algorithm.

```
procedure ADWIN(ε)
    initialize window W
    for each n > 0 do
        W ← W ∪ {x_n}        ▷ add x_n to the head of W
        if ‖μ̂_{W_0} − μ̂_{W_1}‖ ≥ ε for some split of W = W_0·W_1
    then
            drop instances from the tail of W
```

B. Concept Drift Detection

Concept drift happens when the statistical properties of the analyzed dataset abruptly shift in time [22]. Different change-detection algorithms can be applied to identify the times when the probability distribution of a stochastic process or time series changes. In our problem, such a detection has to be performed in an online manner, i.e. without assuming that the statistics of the complete time series are known in advance. The ADWIN algorithm is by design an online change-detection algorithm. In this paper, we consider an additional change-detection algorithm to analyze the considered dataset: the Page-Hinkley test (PHT) [35]. PHT is a standard statistical test for change detection, commonly used in time-series analysis. In a nutshell, this test is a sequential adaptation of a test for an abrupt change of the average of a Gaussian stochastic process, and it allows efficient detection of changes in the usual behavior of a process.

IV. RAL – Stream Learning with Active Reinforcement

RAL relies on reinforcement-learning principles, using rewards and contextual-bandit algorithms [23], as well as prediction uncertainty. The overall idea is summarized in Figure 1. The intuition behind the different reward values is that we attribute a high (positive) reward in case the system behaves as expected, and a low (negative) one otherwise, to penalize it. RAL obtains rewards/penalties as soon as it is asking for ground truth. In a nutshell, it earns a positive reward ρ^+ in case it queries the oracle and would have predicted the wrong label otherwise (i.e. the system made the right decision to ask for the ground truth: the sample is deemed informative) and a penalty ρ^- (i.e. a negative reward) when it asks the oracle even though the underlying classification model would have predicted the correct label (i.e. querying was unnecessary). The rationale for using reinforcement learning is that RAL learns not only based on the queried samples themselves, but also from the usefulness of its decisions. The objective function to maximize is the total reward: $\sum_{i=1}^{n} r_n$, where r_n is the nth reward (ρ^+ or ρ^-) obtained by RAL.

The conceived system additionally makes use of the prediction certainty of the underlying classification model(s). The prediction certainty is defined as the highest posterior classification probability among all possible labels for sample x. More formally, the prediction certainty of a model is equal to $\max_{\hat{y}} P(\hat{y}|x)$, with $\hat{y}$ being one of all the possible labels for x. The rationale behind this design choice is that the model's

Algorithm 2 RAL algorithm.

procedure RAL(x, E, α, θ, ε, η)
  x: sample to treat
  E: set of learners, members of the committee
  α: vector of decision powers of learners in E
  θ: certainty/querying threshold
  ε: threshold for ε-greedy
  η: learning rate
  decisions $\leftarrow \{\}$ ▷ will contain decisions of learners
  for e $\in$ E **do**
    decisions[e] $\leftarrow$ e.askCertainty(x) $< \theta$
  committeeDecision $\leftarrow$ round$\left(\sum_{e \in E} \alpha[e] \cdot \text{decisions}[e]\right)$
  $p \leftarrow \mathcal{U}_{[0,1]}$ ▷ random number drawn from a uniform distribution
  if $p < \varepsilon$ **or** committeeDecision = 1 **then** ▷ ε-scenario or not?
    $y \leftarrow$ acquireLabel(x)
  if committeeDecision = 1 **then**
    $r \leftarrow$ getReward(x, y)
    $\alpha \leftarrow$ updateDecisionPowers(r, E, decisions, committeeDecision, α, η)
    $\theta \leftarrow \min\left\{\theta\left(1+\eta\times\left(1-2^{\frac{r}{\rho^-}}\right)\right),\quad 1\right\}$
function UPDATEDECISIONPOWERS(r, E, decisions, committeeDecision, α, η)
  for e $\in$ E **do**
    if decisions[e] = committeeDecision **then**
      $\alpha[e] \leftarrow \alpha[e] \times \exp(\eta \times r)$ ▷ EXP4
  return $\alpha / \sum_{e \in E} \alpha[e]$ ▷ normalize each value of α
function GETREWARD(x, y)
  return (ρ^- **if** $\hat{y}(x) = y$ **else** ρ^+)

prediction uncertainty is an appropriate proxy for assessing the usefulness of a data point. Combining the reward mechanism with the model's uncertainty allows us to tune the sample-informativeness heuristic to better guide the query decisions.

Also inspired by the bandit literature [31] and to better deal with concept drifts in the data, we implement an ε-greedy policy, which improves the data-space exploration; we sample a uniform-probability distribution, and if this value is below a certain threshold ε, the system queries the oracle, ignoring the decision of RAL's classification models. We refer to this as the *ε-scenario*. This ensures that we have a good chance of detecting potential concept drifts: without this policy, the system could end up being too confident about its predictions, and thus never ask the oracle again, even though its estimations are wrong.

Next, we present the details of a committee or multi-classifier version of RAL, relying on multiple models. Nevertheless, it is very easy to use RAL with a single machine-learning model, even if we do not use it in the evaluations of this paper. We provide some comments on this by the end of the section.

The algorithm behind RAL is summarized in Algorithm 2. Our approach is inspired by contextual bandits [23]. We rely on a set of experts (i.e. different machine-learning models), referred to as a *committee*. Each expert gives its opinion for the sample to consider: should the system ask the oracle for feedback or is the expert confident enough about its prediction? To assess a model's prediction certainty, we rely on a certainty threshold θ: if the model's certainty is below θ, the expert is too uncertain about the prediction to make and thus it advises that RAL asks the ground truth. The query decision of the committee takes into account the opinions of the experts, but also their decision power: if the weighted majority of the experts votes against querying, RAL will rely on the label prediction provided by the committee, used in the form of a voting classifier. The decision power of each expert gets updated such that the experts which agree with the entire committee are obtaining more power in case that particular decision is rewarding, i.e. informative (otherwise, these experts get penalized). These weights are updated through the EXP4 rule [23], with a learning rate η. RAL does not update the decision powers of the different learners in the ε-scenario: the committee did not take the querying decision and thus the weights of the models should not be impacted by this querying action.

The computation of the reward is carried out every time the committee decided to query (i.e. not in the ε-scenario). RAL therefore gets rewarded with $\rho^+ > 0$ when it queried the oracle and asking was informative (i.e. the voting classifier would have predicted the wrong label). Conversely, RAL is penalized with $\rho^- < 0$ if the system used the oracle because the committee decided to do so, even though the underlying classifier would have predicted the correct class.

As an additional step, to ensure that RAL adapts as best as possible to the data stream, we do not only tune the weights of the committee members based on rewards, but also the uncertainty threshold θ, denoted in the remainder of this section as θ_n to stress that it is influenced by the $n-1$ samples observed so far. Again, as for the decision powers, θ_n is not updated in the ε-scenario. The update rule of θ_n we implemented for our tool is written as follows:

$$\theta_n \leftarrow \min\left\{\theta_{n-1}\times\left(1+\eta\times\left(1-2^{\frac{r_n}{\rho^-}}\right)\right),\quad 1\right\}$$

We now detail the reasoning behind the selection of the update policy used by RAL. We are looking for an update rule of the form

$$\theta_n \leftarrow \min\left\{\theta_{n-1}\times\left(1+f(r_n)\right),\quad 1\right\}$$

where $f(r_n) = 1 - \exp\left(a \times r_n\right)$. The threshold should increase slightly when the reward is positive, conversely when the reward is negative. More formally, the update policy should satisfy the following properties:

1 – θ_n should decrease *fast* in case r_n is negative, as this indicates the system queries too often, thus is doing poorly. Therefore, θ_n should be adapted fast to improve RAL's performance.

2 – θ_n should *slightly* increase when r_n is positive, so that the system does not always keep decreasing the threshold and

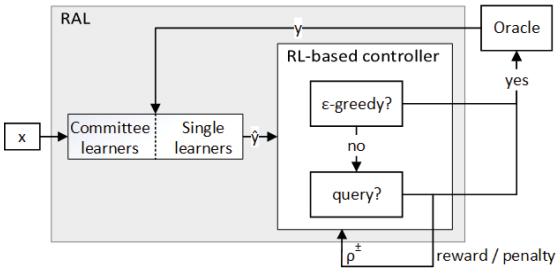


Figure 1: Overall idea of the system.

avoids that θ_n drops to 0. The model was right to ask for more samples, and thus the threshold should be increased. Nevertheless, as the system is currently doing well, we do not want the threshold to be too reactive to the queries.

3 – f **must have two extrema**: a minimum at $\rho^- < 0$ and a maximum at $\rho^+ > 0$.

4 – θ_n **represents a probability**. $\theta_n = 0$ is not acceptable due to the product form of the update policy, thus the values of θ_n must be in the interval $(0, 1]$.

5 – $f(r_n)$ **must be in the interval** $(-1, 1]$ to ensure that θ_n takes values corresponding to a probability. We exclude -1 from the allowed range of values to avoid that θ_n drops to 0.

Properties 1 and 2 lead us to choose the family of functions $f : x \mapsto 1 - \exp(a \times x)$ parameterized by a. Property 5 can be translated into an equation to determine this parameter: $\lim_{r \to \rho^-} f(r) = 1 - \exp(a \times \rho^-) = -1$. After solving this equation, we get $a = \frac{\ln 2}{\rho^-}$. As f is strictly increasing, and because a is nonpositive, f will have a maximum when $r_n = \rho^+$ (thus satisfying property 3). Note that, in order to satisfy property 5, ρ^+ must be chosen such that $f(\rho^+) \leq 1$.

As a final step, we introduce an additional hyper parameter to the update rule, namely the learning rate η. This rate aims at smoothing the evolution of the threshold θ_n, i.e. avoiding that θ_n changes too dramatically with a single query. We thus have the following update rule:

$$\theta_n \leftarrow \min\left\{\theta_{n-1} \times \left(1 + \eta \times \left(1 - 2^{\frac{r_n}{\rho^-}}\right)\right), \quad 1\right\}$$

We restrict the values of η to the range $(0, 1)$. Indeed, we still must satisfy property 5 (a value of 1 would violate this one) and $\eta = 0$ would lead to an nonreactive system, as the threshold would never adapt.

We acknowledge that RAL includes a non-negligible number of hyperparameters which should be well chosen in order to obtain the best results. While we do not have any rule of thumb on how to define exact values, the following guidelines help RAL learn from the streaming data:

1 – The initial value of θ should be set to a high one (i.e. close to 1) when the number of possible labels is low, to avoid that the model is always too certain about its prediction for the encountered samples.

2 – ε should be higher when dealing with more dynamic datasets, to increase the probability of accurately grasping concept drifts; in general, we would advise using values in the range of 1 to 5%.

3 – η should be small to avoid changing the decision powers of the different learners, i.e. α, and θ_n too abruptly; we would advise values below 0.1.

4 – There is no specific range of values for $\rho^\pm$ which works better than others and these values should be picked considering the situation in which RAL is used: if unnecessary queries are a major issue, one should set ρ^- such that its absolute value is much higher than the one of ρ^+.

To conclude, and as we said before, RAL can also easily be used with a single classifier instead of a committee of learners. Transforming the committee version into a single-classifier one is straightforward. In that case, RAL becomes very lightweight and the only element of the system that allows it to efficiently adapt to and learn from the data stream is the variation of the uncertainty threshold θ, by relying on the rewards.

V. Continuous Detection of Network Attacks

To evaluate the performance of the proposed algorithms and adaptation strategies, we consider the detection of diverse types of network attacks in real network-traffic measurements collected at the WIDE backbone network, using the well-known MAWILab dataset for attack labeling. MAWILab is a public collection of 15-minute network-traffic traces [2] captured every day on a backbone link between Japan and the US since 2001. Building on this repository, the MAWILab project uses a combination of four traditional anomaly detectors (PCA, KL, Hough, and Gamma) to partially label the collected traffic.

Given the different nature and goals of ADAM and RAL, we rely on two different evaluation strategies, each of them adapted to the specific challenges tackled. Next, we describe the dataset used in the evaluations, as well as the employed evaluation strategies, along with the obtained results.

A. Data Description

The traffic studied in this paper spans two weeks of packet traces collected in late 2015. From the labeled anomalies and attacks, we specifically focus on those which are detected simultaneously by all four MAWILab detectors. We consider in particular two types of attacks: flooding attacks, and distributed network scans. We train different models to detect each of these attack types separately, thus each detection approach consists of two different binary detectors.

To detect these attacks, we consider a slotted, time-based evaluation. To do so, we split the traffic traces in consecutive time slots of five seconds each, and compute a set of features describing the traffic in each of these slots. In addition, each slot i is assigned a label l_i, consisting of a binary vector $l_i \in \{0, 1\}^2$ which indicates at each position if anomaly of type $j = 1, 2$ is present ($l_{i,j} = 1$) or not ($l_{i,j} = 0$) in the current time slot. We compute a large number n of features describing a time slot, using traditional packet measurements including traffic throughput, packet sizes, IP addresses and ports, transport protocols, flags, etc. The total set accounts for $n = 245$ features, which are computed for every time slot i. Besides using traditional features such as min/avg/max values of some of the input measurements, we also consider

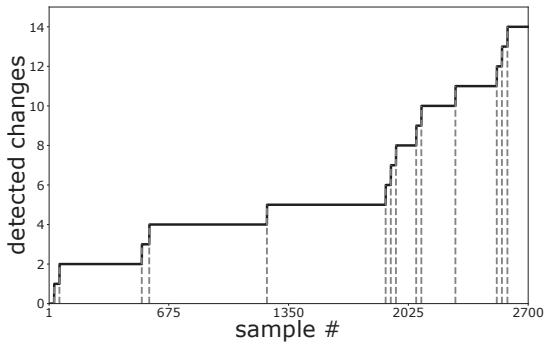


Figure 2: *Page-Hinkley concept-drift detection.* Changes are marked with dashed lines.

the empirical distribution of some of them, sampling it at many different percentiles. This provides much richer information, as the complete distribution is taken into account. We also compute the empirical entropy $H(\cdot)$ of these distributions, reflecting the feature dispersion.

B. ADAM Evaluation Strategy and Performance

A commonly used evaluation scheme within the data-stream-mining domain is the well-known prequential approach. Each instance is first used to test a model, and then to train or update it. Prequential evaluation can be used to measure the accuracy of a model since the start of the evaluation, by keeping in memory the complete history of instances and evaluating the model on each new instance, but it is generally applied using sliding windows or decaying factors – as we do it in ADAM –, which forgets previously seen instances in the model-update process and focuses on those instances in the current sliding window or learning memory. As opposed to more traditional k-fold cross-validation, which is generally used in the evaluation of offline machine-learning models based on k shuffles of the complete dataset, prequential cross-validation works on a single stream of data using only one model: its assessment of the stream-based model tends to be weaker due to this.

To avoid this weakness, we evaluate ADAM following a new strategy to evaluate stream-based algorithms [19], using prequential k-fold cross-validation; this strategy is basically an adaptation of k-fold cross-validation to the streaming setting, and assumes we have k different models derived from the algorithm we want to evaluate, running in parallel. Each time a new sample arrives, it is used for testing one of the k models selected randomly, and is then used for training by all the other models. As evaluation metric, we take the attack-detection accuracy (ACC).

A commonly applied approach to evaluate the performance of stream-based algorithms is to benchmark them against their corresponding offline, batch implementations. Therefore, we use as baseline the results obtained for the corresponding batch-based algorithms, including k-NN, Hoeffding tree (HT-batch), random forest (RF-batch), and SVM. We compute the difference between the batch accuracy and the prequential one to compare the performance of stream algorithms with respect to their batch variants.

We use the MOA machine-learning library [17] to perform the analysis, including both the hyperparameter calibration (using a grid-search procedure) and the model training and evaluation. MOA is specifically designed for stream-based machine-learning approaches.

First of all, we study the variation of the statistical properties of the considered dataset, in particular detecting concept drifts with the Page-Hinkley test. Figure 2 depicts the cumulative number of changes observed in the dataset, as well as the times when those changes are detected. The test detects 14 abrupt changes during the total measurement time span. The frequency of changes significantly increases in the last third of the dataset, with more than 10 changes detected in the last 4 days. Concept drifts occur from modifications of the underlying characteristics of the prediction target.

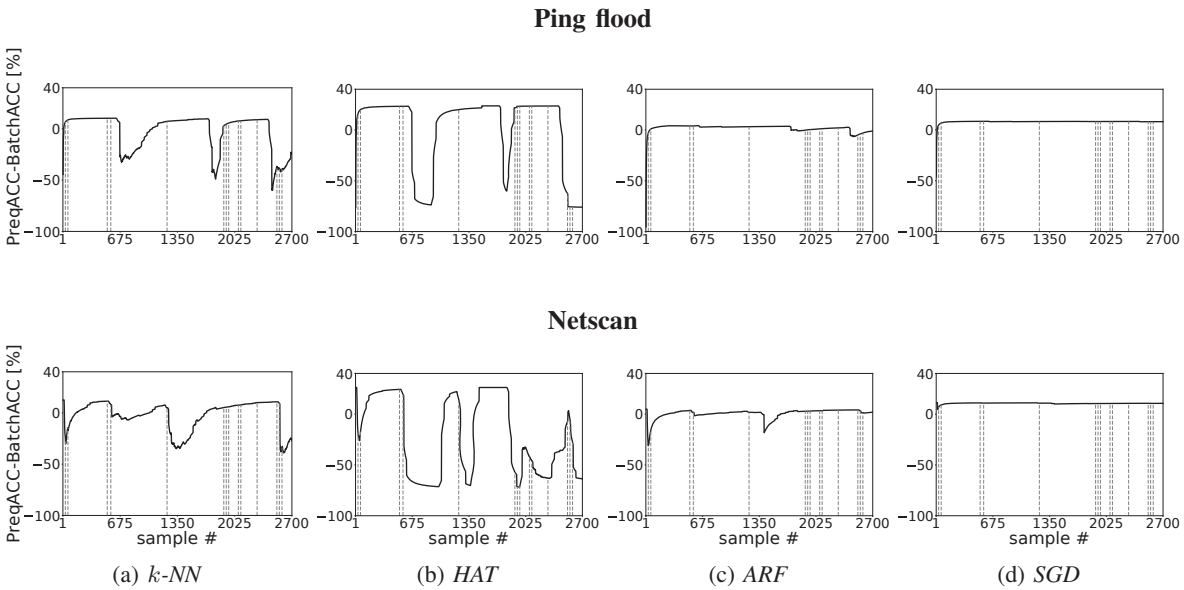
Concept drifts can be used to explain sudden shifts in the performance of algorithms as depicted in Figure 2. For example, the window size of the algorithm changes when concept drifts are detected, and thus could have an important impact on performance.

We evaluate the performance of the learning algorithms using ADAM in two binary-classification scenarios – one for each attack type, resulting in two sets of results for each tested algorithm. Figure 3 reports the performance results for each attack type, considering detection accuracy as the performance metric, and using the batch-algorithm accuracy as baseline. More precisely, we report the difference between the 10-fold prequential and batch accuracy. The prequential CV-performance evaluation shows that both ARF and SGD models rapidly converge to the batch-based accuracy results, with minimum accuracy variations when concept drifts occur. The SGD model performs slightly better, even outperforming the batch-based performance. On the other hand, both the k-NN and HAT models do not exhibit any convergence and results tend to oscillate around the batch-algorithm baseline. In the case of HAT, we can highlight the correlation between the detected concept drifts and the performance variations of the model. Interestingly, the HAT model is the one achieving the highest accuracy of the four models (up to 30% improvement with respect to the baseline), but cannot maintain such a performance constantly in time.

C. RAL Evaluation Strategy and Performance

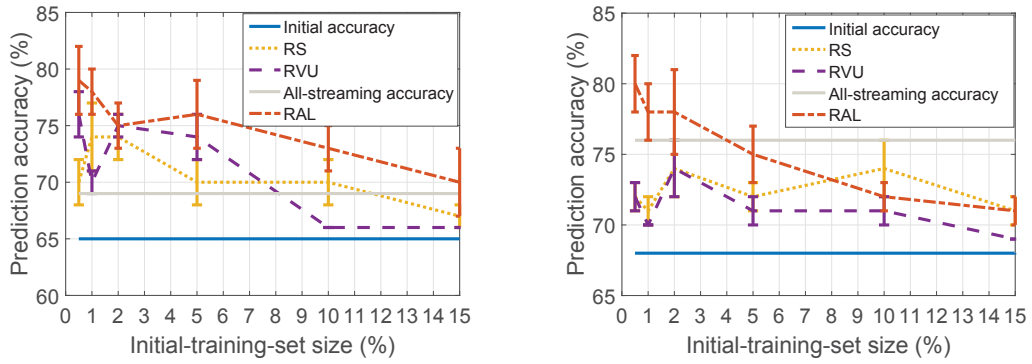
To showcase the performance of RAL, we evaluate and compare it to a state-of-the-art algorithm for stream-based active learning, as well as against a very basic random sampling approach (RS). In particular, we compare RAL to the Randomized Variable Uncertainty (RVU) technique proposed in [33], [34], as this approach also heavily relies on the uncertainty of the underlying machine-learning models to take the querying decisions.

For each benchmarked algorithm, we proceed as follows: first, we subdivide the considered datasets into three consecutive, disjoint parts, i.e. the initial training set, the streaming



(a) *k-NN* (b) *HAT* (c) *ARF* (d) *SGD*

Figure 3: *Prequential 10-fold cross-validation accuracy evaluation.* Diagrams show prequential 10-fold CV results for each algorithm for each attack type. Concept drifts detected by the Page-Hinkley test are marked with dashed lines.



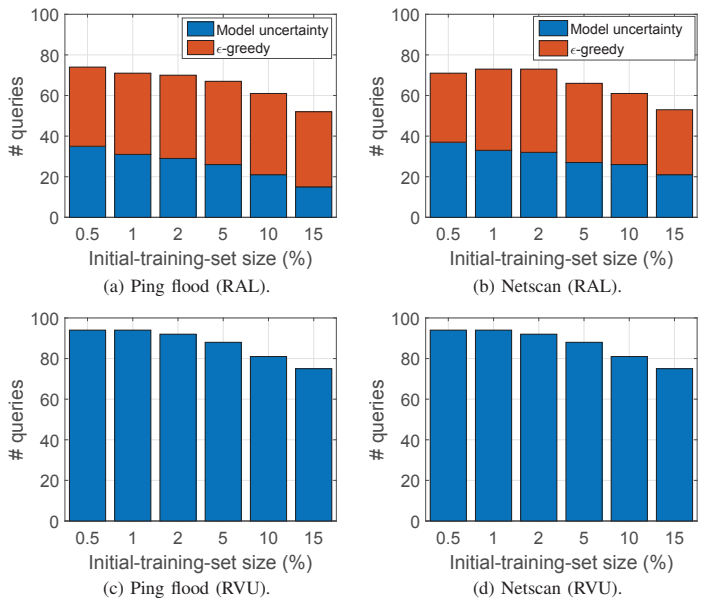
(a) Ping flood. (b) Netscan.

Figure 4: Prediction accuracy for RAL, RVU, and RS. For each of the tested datasets, RAL outperforms both techniques.

data, and the validation set. The validation set consists of the last 30% of the dataset, the initial training set is a variable fraction of the first samples (varying between the first 0.5%, 1%, 2%, 5%, 10%, and 15%), and the streaming part includes all the remaining samples not belonging to the other two subsets. We then train a model on the initial training set and check its accuracy on the validation part – we refer to this as the *initial accuracy*. Next, we run the specific active-learning algorithm on the streaming part and let it pick the samples it decides to learn from. We retrain the model after each new queried label. Finally, we evaluate the final model – trained on the initial training set plus the selected samples –, again on the validation set, and analyze this prediction accuracy – referred to as the *final accuracy*.

In the context of this evaluation, we implement for both RAL and RVU the budget mechanism presented in [33], based on the ratio between the number of queries and the total number of samples observed so far; the system is allowed to issue queries to the oracle as long as this ratio is below a certain threshold, i.e. the budget. For RS, we use a budget indicating the exact number of samples to ask feedback for. For each attack type, we set it to the highest average number of queried samples by either RAL or RVU among all the tests with all the considered initial-training-set sizes.

Similarly to the evaluation of ADAM, all tests are repeated 10 times, and we report both average accuracy and standard errors. For RAL, we indicate the average number of queries performed due to the uncertainty of the underlying model, as well as those issued through the ε-greedy mechanism. For comparison purposes, we also report the average number of queries issued by RVU. We set the values of ρ^+ and ρ^- to 1 and -1, respectively. Based on grid search with the training data in the ranges presented in Section IV, we set θ to 0.9, ε to 2.5%, and η to 0.01. In the case of RVU, we set its parameters based on those recommended in [34]. Finally, we set the budget to 0.05 for both approaches.



(a) Ping flood (RAL). (b) Netscan (RAL).

(c) Ping flood (RVU). (d) Netscan (RVU).

Figure 5: Number of queries issued by RAL and RVU. RAL asks for much fewer samples, yet can achieve a better accuracy than RVU.

Results obtained for the Ping-flood and the Netscan attacks are presented in Figures 4 and 5. The reported all-streaming accuracy refers to the accuracy obtained by the model in case it queries all the samples seen in the stream. In the context of this evaluation, we use the committee version of RAL. More precisely, the committee is a voting classifier composed of a k-NN model with $k = 5$, a decision tree, and a random forest with 10 trees. We use the same voting model for RVU and RS. Figures 4(a) and 4(b) clearly show that RAL outperforms both RVU and RS on average. A striking example is the result for the Netscan detection, where RAL obtains final accuracies which are up to 10 percentage points higher than the ones of RVU and RS for the two smallest initial-training-set sizes. To our surprise, RVU is often outperformed by RS. Finally, the Ping-flood-detection analysis shows that the three approaches often yield a final accuracy higher than the all-streaming one, underlining that learning from the entire data stream does not necessary output the best possible accuracies. This could be explained by the high number of concept drifts in the data.

Last, when it comes to the number of queried samples, we see that RAL queries on average significantly less often than RVU – between 20% and 25% fewer queries –, and a non-negligible part of these queries are due to the model's uncertainty, suggesting that the samples picked by RAL for its learning purposes are wisely chosen. Also note how useful turns out to be the ε-greedy policy, as the additional exploration capability helps better deal with the concept drifts in the data, contributing to the better results showed in Figure 4.

The initial accuracy is constant for the two different MAWILab attack subsets. This is due to the fact that the first 15% of these datasets consist of points with the same label (more precisely, they represent an attack).

One could wonder whether the performance gain by RAL is worth the complexity of the system. Even though the accuracy gain might not be very significant, RAL's querying strategy has additional advantages over the two other techniques. For instance, RS does not take into account the uncertainty of the model nor query usefulness, meaning that there is a risk to miss interesting samples. Indeed, querying the ground truth when the model is uncertain helps discover underexplored regions the model can learn from and RAL is additionally guided by its reward mechanism. In the specific case of the MAWILab dataset, RS would probably miss interesting attack samples, while RAL has a better chance of querying ground truth for these data points and better learn how to detect attacks. Another advantage of RAL over RVU, besides its better results shown above, is that the querying decisions are influenced by the informativeness of all past queries, not only their sheer execution; RVU does not take that information into account at all, and thus there is a risk that RVU queries unnecessary samples too often. This is especially problematic if querying is very expensive or the oracle has only limited availability.

VI. Concluding Remarks

Dynamic and adaptive-memory-based learning seems to be a promising learning strategy to adapt to very dynamic environments, where concept drifts occur often. This is a common scenario when dealing with online network-traffic-monitoring applications. We have introduced and evaluated ADAM and RAL, two stream-based machine-learning approaches to tackle important challenges when dealing with data streams. We have shown that ADAM permits to track transient changes and concept drifts along time. Indeed, using ADAM, adaptive learning algorithms can continuously achieve high detection accuracy over dynamic network data streams, when dynamically adapting their learning pace and memory to changes in the underlying statistics of the samples. We have confirmed that both adaptive random forests and SVM through stochastic gradient descent are better for the studied problem, especially in terms of robustness to concept drifts and convergence of results. We have also introduced RAL, a novel Reinforced stream-based Active-Learning approach to tackle the challenges of stream-based active learning, i.e. selecting the most valuable sequentially incoming samples to reduce the amount of learning data to label, using reinforcement-learning principles. RAL does not only learn from the data stream, but also from the relevance of its own querying decisions. RAL provides a completely different exploration-exploitation trade-off than existing algorithms. Evaluations have shown that RAL provides very promising results, outperforming state-of-the-art techniques, providing higher accuracies with less ground truth. As an additional contribution, we make RAL freely available on GitHub. As a next step, we are currently working on the integration of ADAM and RAL approaches, using the ADWIN adaptation and concept-drift-detection strategy to improve the performance of RAL under dynamic scenarios, as well as by using a dynamic memory-length approach.

REFERENCES

[1] R. Boutaba et al., "A Comprehensive Survey on Machine Learning for Networking: Evolution, Applications and Research Opportunities," in *JISA*, vol. 9, no. 1, 2018.

[2] R. Fontugne et al., "MAWILab: Combining Diverse Anomaly Detectors for Automated Anomaly Labeling and Performance Benchmarking," in *ACM CoNEXT*, 2010.

[3] P. Casas et al., "Network security and anomaly detection with Big-DAMA, a big data analytics framework," in *IEEE CloudNet*, 2017.

[4] V. Chandola et al., "Anomaly Detection: A Survey," in *ACM Computing Surveys*, vol. 41, no. 3, pp. 15:1–15:58, 2009.

[5] M. Ahmed et al., "A survey of network anomaly detection techniques," in *Journal of Network and Computer Applications*, vol. 60, no. C, pp. 19–31, 2016.

[6] W. Zhang et al., "A Survey of Anomaly Detection Methods in Networks," in *IEEE CNMT*, 2009.

[7] J. Vanerio and P. Casas, "Ensemble-learning Approaches for Network Security and Anomaly Detection," in *ACM SIGCOMM Big-DAMA Workshop*, 2017.

[8] P. Casas et al., "GML learning, a generic machine learning model for network measurements analysis," in *IEEE CNSM*, 2017.

[9] P. Casas and J. Vanerio, "Super learning for anomaly detection in cellular networks," in *IEEE WiMob*, 2017.

[10] V. Carela-Español et al., "A streaming flow-based technique for traffic classification applied to 12+1 years of Internet traffic," in *Telecommunication Systems*, vol. 63, no. 2, 2016.

[11] P. M. Domingos and G. Hulten, "Catching Up with the Data: Research Issues in Mining Data Streams," in *DMKD*, 2001.

[12] M. Stonebraker et al., "The 8 requirements of real-time stream processing," in *ACM SIGMOD Record*, vol. 34, no. 4, pp. 42–47, 2005.

[13] G. Hulten et al., *Mining massive data streams*. University of Washington, 2005.

[14] J. Gama et al., "Issues in evaluation of stream learning algorithms," in *SIGKDD*, 2009.

[15] J. Gama et al., "On evaluating stream learning algorithms," in *Machine learning*, vol. 90, no. 3, pp. 317–346, 2013.

[16] T. R. Hoens et al., "Learning from streaming data with concept drift and imbalance: an overview," in *Progress in Artificial Intelligence*, vol. 1, no. 1, pp. 89–101, 2012.

[17] A. Bifet et al., "MOA: Massive Online Analysis," in *Journal of Machine Learning Research*, vol. 11, pp. 1601–1604, 2010.

[18] A. Bifet and R. Gavalda, "Learning from Time-Changing Data with Adaptive Windowing," in *SIAM International Conference on Data Mining*, 2007.

[19] A. Bifet et al., "Efficient Online Evaluation of Big Data Stream Classifiers," in *ACM SIGKDD*, 2015.

[20] P. Domingos and G. Hulten, "Mining high-speed data streams," in *ACM SIGKDD*, 2000.

[21] H. Gomes et al., "Adaptive random forests for evolving data stream classification," in *Machine Learning*, vol. 106, no. 9, pp. 1469–1495, 2017.

[22] J. Gama et al., "A survey on concept drift adaptation," in *ACM CSUR*, 2014.

[23] P. Auer et al., "The Nonstochastic Multiarmed Bandit Problem," in *SIAM Journal on Computing*, vol. 32, no. 1, pp. 48–77, 2002.

[24] Y. Baram et al., "Online Choice of Active Learning Algorithms," in *Journal of Machine Learning Research*, vol. 5, pp. 255–291, 2004.

[25] W.-N. Hsu et al., "Active Learning by Learning," in *AAAI*, 2015.

[26] D. Ienco et al., "Clustering Based Active Learning for Evolving Data Streams," in *International Conference on Discovery Science*, 2013.

[27] B. Krawczyk, "Active and adaptive ensemble learning for online activity recognition from data streams," in *Knowledge-Based Systems*, vol. 138, no. C, pp. 69–78, 2017.

[28] B. Settles, "Active Learning Literature Survey," Technical report, 2010.

[29] L. Song, "Stream-based Online Active Learning in a Contextual Multi-Armed Bandit Framework," *arXiv preprint arXiv:1607.03182*, 2016.

[30] L. Song and J. Xu, "A Contextual Bandit Approach for Stream-Based Active Learning," *arXiv preprint arXiv:1701.06725*, 2017.

[31] C. Watkins,"Learning from Delayed Rewards," PhD thesis, 1989.

[32] W. Xu et al., "Active learning over evolving data streams using paired ensemble framework," in *ICACI*, 2016.

[33] I. Žliobaitė et al., "Active Learning with Evolving Streaming Data," in *ECML PKDD*, 2011.

[34] I. Žliobaitė et al., "Active Learning with Drifting Streaming Data," in *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 1, pp. 27–39, 2014.

[35] E.S. Page, "Continuous Inspection Schemes," in *Biometrika*, vol. 41, pp. 100–115, 1954.