Catch It If You Can: Real-Time Network Anomaly Detection With Low False Alarm Rates

Georgios Kathareios, Andreea Anghel, Ákos Máté, Rolf Clauberg, Mitch Gusat
IBM Research, Zürich
{ios, aan, kos, cla, mig}@zurich.ibm.com

***Abstract*—Unsupervised anomaly detection (AD) has shown promise against the frequently new cyberattacks. But, as anomalies are not always malicious, such systems generate prodigious false alarm rates. The resulting manual validation workload often overwhelms the IT operators: it slows down the system reaction by orders of magnitude and ultimately thwarts its applicability. Therefore, we propose a real-time network AD system that reduces the manual workload by coupling 2 learning stages. The first stage performs adaptive unsupervised AD using a shallow autoencoder. The second stage uses a custom nearest-neighbor classifier to filter the false positives by modeling the manual classification. We implement a prototype for 10-50Gbps speeds and evaluate it with traffic from a national network operator: we achieve 98.5% true and 1.3% false positive rates, while reducing the human intervention rate by 5x.**

***Keywords*—Security, online, anomaly detection, neural nets.**

I. Introduction

There are two prevalent approaches in designing intrusion and anomaly detection (AD) systems: *signature-* and *behavioral-based*. Signature-based detection relies on the existence of a collection of known attack signatures which gets updated every time a new attack is found. The detection is performed by checking if the signature of suspicious traffic matches a signature in the available collection. While such systems excel in detecting known attacks, they generally fail to detect new malware.

Behavioral-based detection is useful in defending against novel malicious behaviors, for which signatures might not be available. This detection relies on machine learning to create profiles for the *normal* network traffic behaviors. The profiles are used to detect anomalies, i.e., traffic with a behavior that diverges significantly from the norm. A merit of this approach is that it can operate without prior knowledge or traffic assumptions, often being unsupervised in nature.

However, there are multiple challenges that purely unsupervised, behavioral-based network AD mechanisms have to face [1], [2]. The most important is their *high false positive rate*. As the AD is unsupervised, it will identify anomalies in any traffic that is sufficiently far from normal. And yet not all the anomalous traffic is malicious or harmful; i.e., also the benign traffic is widely diverse and may often diverge from the norm. Practically today a completely unsupervised detector cannot be trusted to work autonomously - its results may need to be examined by a human operator before any action is taken [3]. This can significantly overload the human component of the network management infrastructure. As such, it is common for network operators to lose their trust in an AD system that creates a high rate of false positives, i.e., a system that increases their workload and elicits unproductive activities.

A second challenge is the fact that attacks may be highly volatile and long-lasting. Window-based markovian methods [3], [4], [5], despite being highly adaptive, encounter a paradox during long-lasting events that also dominate the traffic, e.g., Denial-of-Service (DoS) attacks: Within some time windows the detector perceives the dominant attack traffic as normal, and the rest (i.e., benign traffic) as anomalous. Hence, there is a need for detection models that can learn the long-term characteristics of the network, but also adapt to new malicious behavior.

The main contributions made in this paper are:

1) The design and test of a network AD system that can operate on streams of both encrypted and non-encrypted network packets. Instead of simply reporting the detected anomalies, our system automatically classifies the majority of them as harmful or non-harmful, with only minimal human intervention.
2) The novel combination of a behavioral-based, unsupervised AD first stage with a signature-based, supervised second stage, which reduces the operator's workload.
3) The design and tuning of a real-time scalable two-stage pipeline (Fig. 1). The first stage uses an autoencoder neural network model, while the second stage uses a nearest-neighbor classifier model. Both stages are custom designed for the next generation datacenter networks. We present below the algorithmic design, implementation and tuning of these particular models.

Our objective is to detect the traffic volumetrics outliers that are strong indicators of attacks (flood attempts, scanning attacks, etc.) or miss-configured equipment. The combination of both unsupervised and supervised stages guarantees that: (a) we detect the *novel* potentially harmful traffic that has not been previously encountered, and, (b) we reduce the need for manual examination of anomalous traffic by automatically filtering new anomalies based on the previously identified false positives. We address the network data variability by implementing the AD pipeline in a way that simultaneously: (1) adapts to changes in traffic behavior through online learning, and, (2) retains memory of past behaviors, in contrast to the prevailing markovian approaches.

The rest of the paper is structured as follows. In Section II we describe the data pre-processing stage of our AD system. In Sections III and IV we present the two main stages of

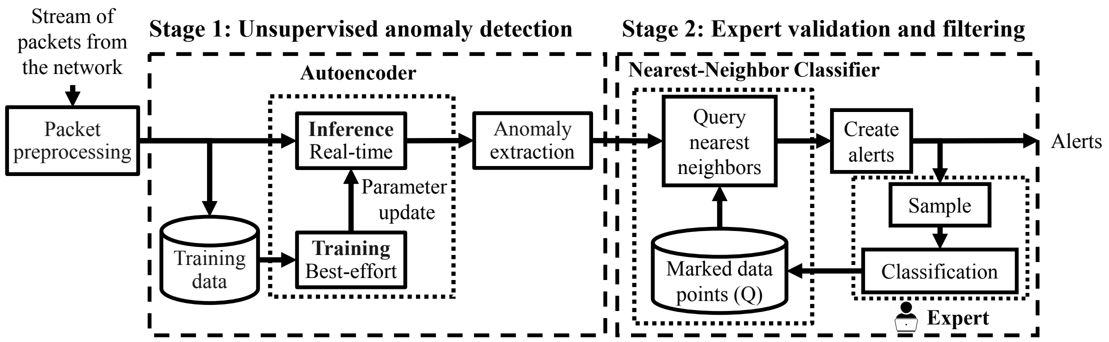

Fig. 1: Overview of the AD system.

the AD pipeline, respectively. We discuss the experimental results using real-world data in Section V. The related work is summarized in Section VI, after which we conclude in Section VII.

II. Data pre-processing

As input we consider streams of raw packets originating directly from a network link. Due to the ubiquity of end-to-end encryption in today's communications, we do not use Deep Packet Inspection, but instead base the detection only on (a) the information included in the Layer 2 to 4 headers of the packets and (b) metrics of the network flows themselves (e.g., packets per second rate).

During pre-processing, for each traffic source s we create *per-source flow aggregations* as the sets of all packets that originate from s and have a timestamp within consecutive, non-overlapping time intervals of length Δt. Δt is the user-defined *aggregation interval*, with a default value of 1 sec. For each flow aggregation we then define a *data point*, a vector of n features computed over it. The features are any metrics that can be extracted from the flow aggregation, categorized as: (a) protocol-specific, e.g., the number of packets with the TCP SYN flag raised, (b) communication-pairs-specific, e.g., ratio of destination-to-source ports, (c) packet-specific, e.g., total number of packets and total bytes in the flow aggregation. For all features that represent counters of packets with a specific property, we also introduce features that represent the ratio of this kind of packets to the total number of packets in the aggregation. The features are normalized in an online manner using their respective exponentially weighted means and standard deviations, and are mapped to the range $(-1,1)$ with a hyperbolic tangent function. We denote as $\mathbf{x}_i$ the i-th normalized data point and as $x_{i,j}$ its j-th feature. The time-series generated from pre-processing the input is:

$$\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \dots) = ((x_{1,1} \dots x_{1,n}), (x_{2,1} \dots x_{2,n}) \dots) \quad (1)$$

III. Stage 1: Unsupervised anomaly detection

The goal of the first stage of the pipeline is to assign to each data point $\mathbf{x}_i \in \mathbf{X}$ an anomaly score $a(\mathbf{x}_i)$: a scalar value that represents the degree to which the data point diverges from the normal behavior. We use the reconstruction error of an autoencoder as an approximation of the anomaly scores.

We selected the autoencoder neural network for a number of merits it presents over other methods. The main advantage of using neural networks is that no assumptions on the distribution of the input data are necessary, as the model is able to discover the most relevant features by itself. Thus, compared to clustering, autoencoders do not depend on the notions of distance or density in the input data. Also, autoencoders that use non-linear encoding and decoding functions have the capacity to learn a non-linear generalization of vanilla PCA [6], and can thus model more complex behaviors. Kernel-based PCA [7], [8] could address the linearity limitations of vanilla PCA. However, as in any kernel-based methods, the selection of the kernel highly depends on the distribution of the input data, which in our case is unknown and non-stationary.

An autoencoder with parameters $\boldsymbol{\theta}$ is a feed-forward neural network that implements an encoding ($f_{\boldsymbol{\theta}}$) and a decoding ($g_{\boldsymbol{\theta}}$) function. Given an input data point $\mathbf{x}_i$, the autoencoder maps it to a code $\mathbf{c}_i$ and then outputs $\mathbf{y}_i$:

$$\mathbf{c}_i = f_{\boldsymbol{\theta}}(\mathbf{x}_i), \;\; \mathbf{y}_i = g_{\boldsymbol{\theta}}(\mathbf{c}_i) = g_{\boldsymbol{\theta}}(f_{\boldsymbol{\theta}}(\mathbf{x}_i)) \quad (2)$$

The output $\mathbf{y}_i$ is the reconstruction of $\mathbf{x}_i$ according to $\mathbf{c}_i$ and $\boldsymbol{\theta}$. Such a network can be trained so as to minimize the difference of the input $\mathbf{x}_i$ and the output $\mathbf{y}_i$ in order to create accurate reconstructions. Therefore, the training phase tries to estimate the set of parameters $\hat{\boldsymbol{\theta}}$ that minimize the mean squared error over some subset $\mathbf{X^b}$ of the set of normalized data points:

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \left(\frac{1}{|\mathbf{X^b}|} \sum_{\mathbf{x}_i \in \mathbf{X^b}} \|g_{\boldsymbol{\theta}}(f_{\boldsymbol{\theta}}(\mathbf{x}_i)) - \mathbf{x}_i\|^2 \right) \quad (3)$$

In order for the autoencoder to model the inherent characteristics of normal traffic behavior, we design it to be *under-complete*. We restrict the dimension of the code $\mathbf{c}_i$ to be less than n, forcing the model to perform dimensionality reduction on the input [6]. In the opposite case, the autoencoder would learn the identity function and would provide no interesting information. Learning an undercomplete representation forces the autoencoder to capture the most salient features of the training data, which, in the context of network traffic, is the form of the traffic most commonly encountered. Thus, the model will be able to accurately reconstruct data points that are close to the norm, and will have a high reconstruction error on anomalous data points. We take advantage of this property and calculate the anomaly score as the reconstruction error:

$$a(\mathbf{x_i}) = ||\mathbf{g}_{\hat{\boldsymbol{\theta}}}\left(\mathbf{f}_{\hat{\boldsymbol{\theta}}}\left(\mathbf{x_i}\right)\right) - \mathbf{x_i}||^{\mathbf{2}} \quad (4)$$

Algorithm 1 Batch selection for some t, a, b, c

```
1: $k \leftarrow 0$
2: $\mathbf{X^b} \leftarrow \{\}$
3: for $i \in [0,b)$ do
4:     $\mathbf{X^b} \leftarrow \mathbf{X^b} \cup \{\mathbf{x}_{(t-a(k+1))}, \ldots, \mathbf{x}_{t-ak}\}$
5:     $k \leftarrow k + 1 + (i \div c)$        ▷ Integer division
6: return $\mathbf{X^b}$
```

It is expected that attacks can appear in a very sudden manner. Therefore, the anomaly score for each data point must be computed in near real-time. To do so, we separate the slower training from the faster score inference in two different instances of the model that run in parallel. The first instance is used for training and uses error backpropagation and gradient-based learning (e.g., Adagrad [10]). Every time $\boldsymbol{\theta}$ is updated after the error propagation of a single batch, $\boldsymbol{\theta}$ is communicated to the second instance of the model. There, $\boldsymbol{\theta}$ is used for real-time inference on the input time series until the next update. The updates of the model parameters essentially adapt the view of the model on what is considered normal traffic. As we do not expect the notion of normal to change much over time, there is no need for model updates after each data point. Instead, by using an update interval of a few seconds, we remove the slow training from the critical path of the pipeline, without any hindrance to the model in detecting novel anomalies in real time.

The size of each training batch $\mathbf{X^b}$ and the points it contains affect (a) the rate of parameter updates and (b) how fast the model adapts to more recent input. A simple yet efficient way to create the batches that worked well for us involves grouping the points in blocks of size a. We use batches with size equal to b blocks ($a \cdot b$ data points). The batch is calculated with the heuristic shown in Algorithm 1, where $\mathbf{x}_t$ is the most recent data point and c is some non-negative integer. This heuristic essentially adds to the batch a sample of data points from a large range[1] but with a distribution skewed towards $\mathbf{x}_t$. The parameters a and b control the batch size and together with c they control how much past information is included in the batch. Thus, each update of $\boldsymbol{\theta}$ adapts the model to the newest data points, but retains characteristics of past data points.

As final step of the first stage, the anomaly scores are normalized in a manner similar to the data point features and the anomalies are extracted. The anomaly extraction step classifies as anomalies the data points with normalized scores above a threshold Ω, creating the time series $\mathbf{A}$ of anomalous points:

$$\mathbf{A} = (\ldots, \mathbf{x}_i, \ldots), \forall \mathbf{x}_i \in \mathbf{X} \text{ where } a'(\mathbf{x}_i) > \Omega \qquad (5)$$

where $a'(\mathbf{x}_i)$ is the normalized value of $a(\mathbf{x}_i)$.

IV. Stage 2: Supervised anomaly classification

Unsupervised AD cannot operate autonomously because not all anomalies constitute malicious behavior. In such systems the results have to be examined and validated by an expert before an action is taken. For the second stage of the pipeline, we create a model based on the decisions of the expert on a small sample of $\mathbf{A}$. The model then accurately decides on behalf of the expert on the majority of the data points. This partial automation (1) reduces the number of alerts the expert receives, and, therefore, the number of false positives that need to be validated, and, (2) allows the system to ingest a higher rate of incoming data points, as the bottleneck of manual validation is greatly reduced.

We assume that the expert has the ability to perform binary classification for each data point in $\mathbf{A}$ based on the values of the features $\mathbf{x}_i$ and possibly the corresponding anomaly score $a'(\mathbf{x}_i)$. The expert performs the following mapping:

$$f_E : \mathbf{A} \rightarrow \{\texttt{threat}, \texttt{non}-\texttt{threat}\} \qquad (6)$$

In order to avoid the high cost of missclassification, for the model we add a third possible label. This way, the data points for which there is a high degree of uncertainty can be classified neither as $\texttt{threat}$, nor as $\texttt{non}-\texttt{threat}$. The model thus performs the following mapping:

$$f_M : \mathbf{A} \rightarrow \{\texttt{threat}, \texttt{non}-\texttt{threat}, \texttt{don't know}\} \qquad (7)$$

On the one hand, f_M must produce the same classification as the expert would as often as possible. On the other hand, when there is a high degree of uncertainty, the label $\texttt{don't know}$ is preferred over missclassifying.

For modeling the classification we use a nearest-neighbor classifier (NNC). As we expect a large degree of data locality in the labeled data points, e.g. points that are part of the same attack would be very similar, the selection of distance-based models like the NNC follows naturally. A *threshold-based nearest-neighbor classifier* (tNNC) is used instead of a k-nearest-neighbor classifier because we need the system to consider all neighboring anomalies instead of just the k nearest ones.

The classification procedure takes place as follows. First, the expert is presented with $\mathbf{A}$. However, due to the potentially high rate of the time series, the expert can only process anomalies by sampling $\mathbf{A}$. The time series of anomalies that is created by sampling is denoted by $s(\mathbf{A})$. Next, each anomalous data point $\mathbf{a}$ of $s(\mathbf{A})$ is classified by the expert, and stored along with its label $f_E(\mathbf{a})$ in a FIFO queue Q with maximum size $|Q|_{max}$. Based on the current state of Q, the tNNC will first calculate for each anomaly $\mathbf{a} \in \mathbf{A}$:

$$\mathbf{N}_T^Q(\mathbf{a}) = \{q, q \in Q \text{ and } d(\mathbf{a}, q) < T\} \qquad (8)$$

which is the set of anomalies in Q within the T-neighborhood of $\mathbf{a}$, for some distance metric d (e.g., Euclidian) and threshold T. It will also measure the number of threat and non-threat neighbors in $\mathbf{N}_T^Q(\mathbf{a})$, $t_T^Q(\mathbf{a})$ and $nt_T^Q(\mathbf{a})$, respectively. The classification is then computed as follows:

$$f_M(\mathbf{a}) = \begin{cases} \texttt{threat} & \text{iff } \frac{t_T^Q(\mathbf{a})}{|\mathbf{N}_T^Q(\mathbf{a})|} > C \\ \texttt{non}-\texttt{threat} & \text{iff } \frac{nt_T^Q(\mathbf{a})}{|\mathbf{N}_T^Q(\mathbf{a})|} > C \\ \texttt{don't know} & \text{else} \end{cases} \qquad (9)$$

[1]The range can be calculated as : $\left(\mathbf{x}_{\frac{ab}{2}\left(\frac{b}{c}+1\right)}, \ldots, \mathbf{x}_t\right)$.

where C is a threshold that controls the confidence that the tNNC needs before labeling with `don't know`.

Overall, by introducing the second stage of the pipeline the expert only needs to process the sample of anomalies plus the anomalies that the tNNC cannot classify. Therefore, with appropriate values for the $s(\mathbf{A})$ rate, $|Q|_{max}$ and C, we can tune the accuracy of the tNNC so that the rate of anomalies that need to be processed by the expert becomes much lower than the rate of all the identified anomalies that would need to be processed otherwise.

V. Experimental Results

The AD pipeline was tested on real-world data from the network of the National Information Infrastructure Development Institute (NIIFI) of Hungary. Our data consists of packet captures from a 10 Gbps link transferring general Internet traffic. The capture covers 3.5 hours of traffic and corresponds to a time series $\mathbf{X}$ of 59,750,000 data points. The capture also includes a small-scale UDP flood attack, during which, 42 external sources attempt to flood a specific destination in the network, by sending high rates of minimally-sized UDP datagrams.

Besides the flood attack, within the same data, the detector identified a number of anomalies that had not been previously detected by the network operators. A number of them showed malicious behavior (see Table I).

A. Evaluation of Stage 1

For our experiments, we used input vectors with $n = 27$ features, and a 5-layer autoencoder, with layer sizes of $27, 20, 10, 20$, and 27. The neural network uses the hyperbolic tangent as the activation function and batch normalization [11]. As the model is trained on data streams and not on a static dataset, there is no danger of overfitting, thus no regularization is required.

For training, we used batches of 2.5M data points ($a = 50000$, $b = 50$, $c = 10$ in Alg. 1). Each iteration of the parameter update takes 8.1 seconds when training takes place in the GPU and 20.5 seconds when performed on the CPU. Fig. 2 shows the mean train error and test error as the model parameters are updated with each batch. As test error for batch i we calculate the mean reconstruction error for all data points that are processed between the i-th and $i+1$-th parameter updates. As shown, after about 50 parameter updates has learned an initial representation of the normal traffic and from that point onward it is able to adapt and keep the low level of test error.

Fig. 3 shows the normalized anomaly scores $a'(\mathbf{x}_i)$ for all data points in a single block. The vast majority of anomaly scores are assigned small values, while the spikes in the figure denote the anomalies included in the block. By varying the threshold value Ω, the amount of data points that are classified as anomalies, and subsequently the number of traffic sources that are detected as anomalous, vary accordingly (see Table II).

To evaluate the correctness of the autoencoder detection, we compared the results of the first stage of the pipeline with an offline PCA-based outlier detection method [12]. For this we considered as input a matrix of the total number of data points. Each feature was mean subtracted and normalized by dividing it by the standard deviation. Using PCA, we calculated the principal components v_j and the associated variance λ_j of each component. As anomaly score we used the Hotelling's T^2 score [12] of each data point x_i:

$$T^2(x_i) = \sum_{j=1}^{n} \left(\frac{|x_i \cdot v_j|}{\lambda_i} \right)^2 \tag{10}$$

These anomaly scores follow an F-distribution and we classify the points that belong to some top percentile as anomalies.

Fig. 4 shows for each of the two methods, how many of the anomalies they identify are also identified by the other method. For $\Omega = 5$, almost all the anomalies identified by the autoencoder belong in the top 5% of PCA results. Conversely, 92% of the scores with the highest 1% T^2 scores are identified by the autoencoder with $\Omega = 3$. Thus, the autoencoder-based streaming methodology and the PCA-based offline method assign high scores predominantly to the same data points but sometimes rank them differently, i.e., place them in different top percentiles.

To quantify the extent to which the first stage produces false positives, we further classified the anomalous sources manually for the case of $\Omega = 7$. We show these results in Table I. Anomaly types with ids $1, 2, 4, 6, 7, 9$, and 10 can be indicators of malicious traffic, but the data points of anomaly type 3 may belong to application servers. Furthermore, from the number of anomalous sources of Table II that are identified over the span of the 3.5 hours of traffic, we can see that on average, 33 to 111 anomalous sources are detected per minute, depending on the value of Ω. This means that real-time manual classification would be very challenging for the human operator. These observations show why there is a need to deploy the second stage of the pipeline.

TABLE I: Anomalies manually identified after stage 1, $\Omega = 7$.

id	Anomaly type	% of anomalous sources
1	High percentage of packets with SYN flag (> 60%)	72%
2	Low ratio of destination to source ports (< 1 : 4)	59%
3	High number of destination addresses (> 100)	49.9%
4	Presence of fragmented packets	31.1%
5	High percentage of packets with PSH flag (> 40%)	10.3%
6	High percentage of packets with RST flag (> 30%)	9.3%
7	High percentage of ICMP packets (> 60%)	3.9%
8	High number of destination ports (> 20)	2.5%
9	High percentage of ICMP echo requests (> 60%)	1.2%
10	Sources of the UDP flood	0.5%
11	Nothing of the above	0.3%

TABLE II: Number of anomalous data points and sources (and corresponding % of the total) with different Ω values.

Ω	Num. of anomalous sources	Num. of anomalous data points
2	23356 (1.01%)	498303 (0.83%)
3	16211 (0.70%)	290933 (0.49%)
5	9520 (0.41%)	170919 (0.29%)
7	6976 (0.30%)	117561 (0.20%)

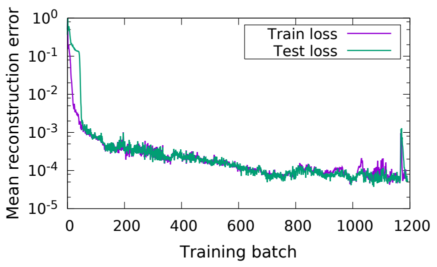

Fig. 2: Autoencoder learning curves.

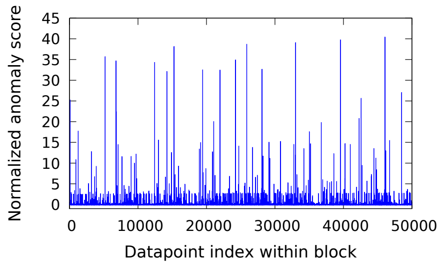

Fig. 3: Anomaly scores (single block).

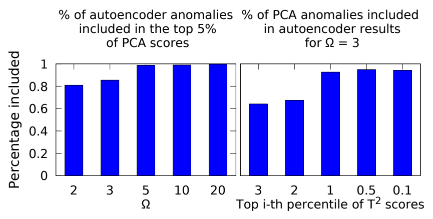

Fig. 4: Offline PCA vs 1st stage.

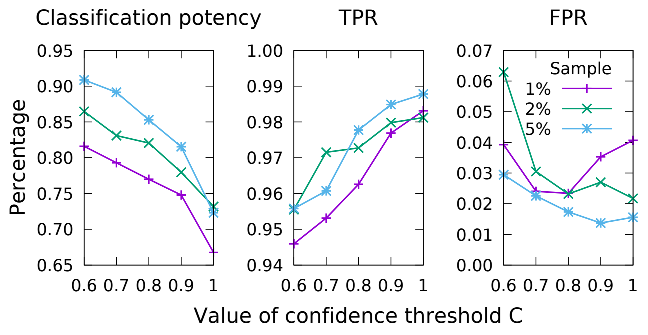

Fig. 5: Classification potency, true and false positive rates for the tNNC with various sample sizes and values for C.

B. Evaluation of Stage 2

For our experiments, we set $|Q|_{max} = 2000$. The amount of available captured data did not allow for tests with larger queue sizes, as the queue would never be full. As n also has a rather low value (27), we opted for a simple exhaustive nearest-neighbor query algorithm. The algorithm compares the distance of the queried point with all points in Q and returns the ones that have a distance within $T = 0.5$. The expert analyzes a sample of the time series of the anomalies with sampling rates that ensure that $s(\mathbf{A})$ has a rate equal to 1%, 2% or 5% of the rate of $\mathbf{A}$.

We evaluate the performance of the tNNC using three metrics: *classification potency*, *true positive rate* (TPR), and *false positive rate* (FPR). We define *classification potency* as the percentage of data points that the tNNC classifies as $\texttt{threat}$ or $\texttt{non} - \texttt{threat}$. TPR and FPR are calculated for the same data points. Fig. 5 shows the results for multiple values of the confidence threshold C and $\Omega = 7$. As expected, with higher C values, the classification potency drops, as there are more cases where the model does not find enough neighbors of the same label with which to classify a data point. For the same reason, TPR increases with higher values of C, as the model only classifies when it has high confidence. FPR generally decreases for the same reason, except for sample size 1%, where the data points in Q are not enough to accurately model the behavior of the expert. In general, the results show that all three metrics improve as the sample size increases.

In Fig. 6 we show the percentage of anomalies that need classification with and without the second stage. For the later case, these anomalies correspond to the sum of the sample $s(\mathbf{A})$ and $\texttt{don't know}$ labels. The results show that as the size of $s(\mathbf{A})$ increases, the expert needs to classify a smaller percentage of anomalies, compared to what would be required if the second stage was not present. Also, with a 5% sample, the false alarms that the expert processes falls from 33.7% to just 8.9% of all anomalies. Therefore, larger sample sizes only benefit the overall system, as with less effort from the expert, stage 2 provides better TPR and FPR.

All in all, these results showcase the extensive benefits of adding the second stage of the pipeline. With an indicative sample size of 5% and a C value of 0.9, the amount of data points that the expert needs to validate drops to just 20.2% of the original size, while we can still accurately model the behavior of the expert with 98.5% TPR and only 1.3% FPR.

C. Timing measurements

A major concern in the design of any AD system is the execution performance, because the system should be able to ingest information from high-speed network links with large number of traffic sources. Table III presents the maximum processing performance of each of the pipeline stages in terms of data points per second. To put things into perspective, the traffic we examined corresponds on average to approximately 4,700 data points generated per second. Therefore, the pipeline has the capacity to ingest a link with up to $4\times$ the rate, or process in real time data points that correspond to a flow aggregation of 0.25 seconds.

As a final experiment, Fig. 7 shows the detection lag for the 42 sources of the UDP flood attack for multiple Ω values. Each point in the figure denotes the percentage of all attackers detected with the corresponding lag. The right-most point of each line corresponds to the total percentage of attackers detected with that Ω value. The results show that most of the attackers are already detected within 1 second, which is equal to Δt, the minimum detection latency possible. For Ω values 3, 5, and 7, all detected attackers are found within just a few seconds from the beginning of the attack. The 10% of attackers that can only be identified when Ω equals 2, corresponds to attackers that have significantly lower rates and are comparable to normal traffic. Thus, on the one hand, they are more difficult to detect, but on the other hand, they do not have a noticeable effect on the network.

TABLE III: Processing speed per stage

Stage	Data points per second
Pre-processing	42909 (before the attack) 26670 (during the attack)
Stage 1	19588
Stage 2	29031

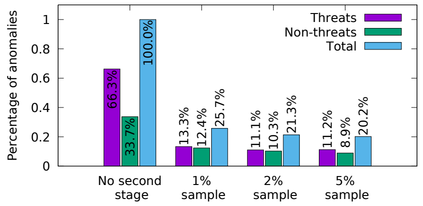

Fig. 6: % of anomalous data points that need expert classification with and without the 2nd stage ($C = 0.9$).

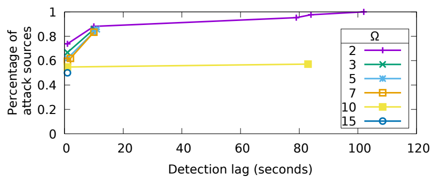

Fig. 7: Time interval between the start of the attack sources' transmission and their detection for the 42 UDP flood sources.

VI. Related work

Many unsupervised network AD methods rely on clustering algorithms [3], [4], [15] or linear PCA [16], [17], [5]. In general, clustering-based methods require making assumptions on the distribution of the input data in order to select optimal distance or density metrics, which are generally unknown or highly-variable in the context of network traffic. Non-linear PCA suffers from the same drawback in kernel selection, and linear PCA may not be expressive enough at times. Instead we opted for a non-linear model (autoencoder) that can model more complex traffic behaviors than vanilla PCA. Also with a neural network, there is no need of prior assumptions on the traffic, as the model is able to discover the most relevant features by itself. Autoencoders have also been used in the past for network intrusion detection [18], [19].

The aforementioned methods do not address the problems of purely unsupervised detection, nor the further examination of results that is needed. In contrast, Veeramachaneni et al. [20] combine unsupervised and supervised models to address this problem, but focus on web and firewall logs, while our approach focuses on low-level network packet streams. Furthermore, Ippoliti and Zhou [21] also use operator feedback to update their one-class SVM-based AD model. To adapt to new traffic behaviors, their model needs to be updated using operator feedback. In contrast, our first stage, which performs the bulk of the anomaly identification, automatically adapts to new traffic without any need for human intervention.

VII. Conclusion

To address the important problem of high false alarm rates commonly encountered in unsupervised systems, we proposed an adaptive, online network AD system targeted to today's high-speed networks. Our system can identify novel malicious traffic similarly to purely unsupervised methods, but requires significantly less manual result examination. The system combines an unsupervised stage that detects novel anomalous behavior, with a supervised stage that models the expert knowledge to filter out false alarms, based on an autoencoder and a nearest-neighbor classifier, respectively. Our experiments on real-world traffic show that the pipeline is able to detect the same anomalies as an offline anomaly detector despite its online mode of operation. Furthermore, it reduces the need for manual anomaly examination by almost 80%, while being able to automatically classify anomalous traffic as malicious with 98.5% true and 1.3% false positive rates.

Acknowledgment

We would like to thank Pál Varga and Péter Orosz for their help and feedback. This research is partly supported by the EU H2020 research and innovation programme under the grant agreement 644960.

References

[1] R. Sommer *et al.*, "Outside the closed world: On using machine learning for network intrusion detection," in *Security and Privacy (SP), 2010 IEEE Symposium on*, 2010, pp. 305–316.
[2] C. Gates *et al.*, "Challenging the anomaly detection paradigm: a provocative discussion," in *Proceedings of the 2006 workshop on New security paradigms*. ACM, 2006, pp. 21–29.
[3] J. Dromard *et al.*, "Online and scalable unsupervised network anomaly detection method," *IEEE Transactions on Network and Service Management*, vol. 14, no. 1, pp. 34–47, 2017.
[4] P. Casas *et al.*, "Unada: Unsupervised network anomaly detection using sub-space outliers ranking," *Networking 2011*, pp. 40–51, 2011.
[5] Y. Liu *et al.*, "Sketch-based streaming pca algorithm for network-wide traffic anomaly detection," in *Distributed Computing Systems (ICDCS), IEEE 30th International Conference on*, 2010, pp. 807–816.
[6] I. Goodfellow *et al.*, *Deep learning*. MIT Press, 2016.
[7] M. E. Tipping, "Sparse kernel principal component analysis," in *Advances in neural information processing systems*, 2001, pp. 633–639.
[8] M. Ghashami *et al.*, "Streaming kernel principal component analysis," in *Artificial Intelligence and Statistics*, 2016, pp. 1365–1374.
[9] Y. A. LeCun *et al.*, "Efficient backprop," in *Neural networks: Tricks of the trade*. Springer, 2012, pp. 9–48.
[10] J. Duchi *et al.*, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.
[11] S. Ioffe *et al.*, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *preprint arXiv:1502.03167*, 2015.
[12] C. Aggarwal, "Outlier analysis," in *Data mining*. Springer'15, pp. 237–.
[13] H. Samet, *Foundations of multidimensional and metric data structures*. Morgan Kaufmann, 2006.
[14] M. Datar *et al.*, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proceedings of the 28th ACM SoCG'04*, pp. 253–262.
[15] A. Lakhina *et al.*, "Mining anomalies using traffic feature distributions," in *ACM SIGCOMM'05*, vol. 35, no. 4, pp. 217–228.
[16] ——, "Diagnosing network-wide traffic anomalies," in *ACM SIGCOMM'04*, vol. 34, no. 4, pp. 219–230.
[17] L. Huang *et al.*, "In-network PCA and anomaly detection," in *Advances in Neural Information Processing Systems*, 2007, pp. 617–624.
[18] S. Hawkins *et al.*, "Outlier detection using replicator neural networks," in *DaWaK*, vol. 2454. Springer, 2002, pp. 170–180.
[19] G. Williams *et al.*, "A comparative study of RNN for outlier detection in data mining," in *IEEE ICDM'02*, pp. 709–.
[20] K. Veeramachaneni *et al.*, "AIˆ 2: training a big data machine to defend," in *IEEE International Conference on Big Data Security on Cloud (BigDataSecurity)*. IEEE, 2016, pp. 49–54.
[21] D. Ippoliti *et al.*, "Online adaptive anomaly detection for augmented network flows," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 11, no. 3, p. 17, 2016.