

Online Self-Supervised Deep Learning for Intrusion Detection Systems

Mert Nakip and Erol Gelenbe *Fellow, IEEE*

Abstract—This paper proposes a novel Self-Supervised Intrusion Detection (SSID) framework, which enables a fully online Deep Learning (DL) based Intrusion Detection System (IDS) that requires no human intervention or prior off-line learning. The proposed framework analyzes and labels incoming traffic packets based only on the decisions of the IDS itself using an Auto-Associative Deep Random Neural Network, and on an online estimate of its statistically measured trustworthiness. The SSID framework enables IDS to adapt rapidly to time-varying characteristics of the network traffic, and eliminates the need for offline data collection. This approach avoids human errors in data labeling, and human labor and computational costs of model training and data collection. The approach is experimentally evaluated on public datasets and compared with well-known machine learning and deep learning models, showing that this SSID framework is very useful and advantageous as an accurate and online learning DL-based IDS for IoT systems.

Index Terms—Self-Supervised Learning, Intrusion Detection, Deep Learning, Internet of Things, Random Neural Network (RNN), Auto-Associative Deep RNN, Botnet Attacks

I. INTRODUCTION

Botnet attacks can lead to thousands of infected devices [1] compromising the devices of victims and turning them into “bots” via malware [2], which in turn cause Distributed Denial-of-Service (DDoS) attacks. The *malicious* bots, i.e. compromised devices, can generate fraud information, cause data leaks, and spread malware. It is reported that 27.7% of all global website traffic in 2021 was generated by bots with malicious intent, and is growing with a 7.3% increase reported between 2018 and 2021 [3].

Botnet attacks severely challenge resource-constrained devices and Internet of Things (IoT) networks [4], as an attack propagates over the victim network increasing network congestion, power consumption, and processor and memory usage of IoT devices over time. Therefore, it is crucial to

detect malicious network traffic and identify compromised IoT devices during an ongoing Botnet attack. While detecting malicious traffic allows reactive actions to alleviate the effects of the attack and stop it, identifying compromised IoT devices paves the way for preventive actions against the spread of malware and Botnet attack.

On the other hand, as the majority (approximately 52%) of IoT connections are to low cost and low maintenance devices deployed in massive IoT networks [5], developing and implementing complex and advanced security methods is challenging as well. To this end, early research [6] developed various types of lightweight Machine Learning (ML)-based Intrusion Detection Systems (IDS) –especially anomaly detecting IDS (anomaly-based IDS)– for IoT networks, showing that anomaly-based IDS is very promising in detecting zero-day attacks based on unknown intrusions that often target vulnerable devices and networks.

Since the decisions of anomaly-based IDS are highly dependent on the characteristics of the normal traffic used for parameter optimization (i.e. learning), accurate decisions become more difficult when the normal behaviour of network traffic changes over time due to both internal and external influences. For example, new device(s) may be added to the IoT network causing a considerable change in aggregated normal network traffic and an increased false positive alarms.

Therefore, anomaly-based IDS could greatly benefit from the ability to adapt in real time to time-varying characteristics of network traffic, ideally through sequential online learning [7], [8]. However, the effectiveness of completely online learning, even for lightweight ML-based IDS, is often limited by two main factors: 1) A sufficient amount of collected and labeled traffic data is not always accessible for every system intended to be secured by the IDS. 2) Online parameter updates are occasionally performed in parallel with intrusion detection at fixed or variable time intervals. Frequent online updates, i.e. short time intervals, result in high computational resource consumption for only minor or no performance gain per update. In contrast, infrequent updates, i.e. long time intervals, may have difficulty adapting to changes in normal traffic, resulting in poor IDS performance. Hence, each time interval needs to be carefully selected, taking into account the current state of the IDS and the actual behaviour of normal network traffic.

In this paper, in order to enable completely online learning of IDS parameters, a novel fully online Self-Supervised Intrusion Detection (SSID) framework is proposed. SSID learns from arriving traffic packets, measures the trustworthiness of the IDS, including its generalization ability and accuracy on

This research has been supported in part by the European Commission H2020 Program through the IoTAC Research and Innovation Action under Grant Agreement No. 952684 and by the European Commission Horizon Europe – the Framework Programme for Research and Innovation (2021-2027) DOSS Project under Grant Agreement No: 101120270.

M. Nakip and E. Gelenbe are with Institute of Theoretical and Applied Informatics, Polish Academy of Sciences (PAN), Gliwice, Poland (e-mails: mnakip@iitis.pl, and seg@iitis.pl)

E. Gelenbe is also with Lab. I3S, Université Côte d’Azur, Nice, France, and Yaşar University, Izmir, Turkey

This preprint is accepted for publication at IEEE Transactions on Information Forensics and Security, DOI: 10.1109/TIFS.2024.3402148. © 2024 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

traffic packets it uses for learning. It can then decide when to update the neural weights via its learning algorithm, keeping itself up-to-date with a high intrusion detection accuracy.

The SSID framework can be used with any anomaly-based IDS that requires parameter optimization, providing fully online self-supervised learning of parameters in parallel with real-time detection requiring no human intervention. It also eliminates the need for labeled or unlabeled offline data collection, and offline training or parameter optimization. Therefore, the proposed framework contrasts sharply with much of existing work [9]–[16] that has implemented self-supervised learning for intrusion detection, often utilizing offline (small-sized) labeled or unlabeled training data and pseudo-labeling. Accordingly, as its main advantages, the SSID framework

- Enables IDS to easily adapt time varying characteristics of the network traffic,
- Eliminates the need for offline data collection,
- Prevents human errors in data labeling (online or offline), and
- Avoids human labor and computational costs for model training and data collection through prior experiments.

We also implement our SSID framework for a Deep Random Neural Network (DRNN)-based IDS that analyzes high-level network traffic metrics extracted from packet header information and learns those metrics calculated for only normal benign traffic. We evaluate the performance of the SSID framework for two tasks, malicious traffic detection and compromised device identification on Kitsune [17] and Bot-IoT [18] datasets. The results revealed that IDS trained under the SSID framework achieve considerably high performance compared to the same IDS with offline and quasi-online (incremental and sequential) learning. Meanwhile, the IDS trained under SSID requires no offline dataset, external parameter optimization or human intervention.

The remainder of this paper is organized as follows: Section II reviews the related work on intrusion detection. Section III presents the overview of the IDS used in this work as well as the detection and learning processes. Section IV proposes the novel SSID framework and present the methodology enabling the self-supervised learning for IDS. Section V evaluates the SSID framework for malicious traffic detection and compromised device identification on public datasets, and compares its performance against the state-of-the-art methods. Finally, Section VI summarizes this paper and provides some insights for the future work. Note that the definitions of abbreviations and the symbols appear in this paper are respectively listed in Table III and Table IV in Appendix.

II. RELATED WORK

We now briefly review recent related work on intrusion detection in three categories of the work that: 1) detect malicious traffic during Botnet-based DDoS (in short Botnet) attacks, 2) identifies compromised network nodes, and 3) performs self-supervised learning for intrusion detection.

A. DDoS Botnet Attack Detection

In [19], Tuan et al. conducted a comparative study for performance evaluation of ML methods aiming to classify Botnet attack traffic. In this work, the authors evaluated the performances of Support Vector Machine (SVM), MLP, Decision Tree (DT), Naive Bayes (NB), and unsupervised ML methods (such as K-means clustering) on two datasets (including KDD'99) revealing that unsupervised ML methods achieve the best performance with 98% accuracy. In [20], Shao et al. created an ensemble of Hoeffding Tree and Random Forest (RF) models with online learning using both normal and attack traffic. In [21], Shafiq et al. developed a feature selection technique as a preprocessing algorithm for an ML-based botnet attack detector. This algorithm ranks features according to their Pearson correlation coefficients and greedily maximizes the detector's performance with respect to area under Receiver Operating Characteristic (ROC) curve in the Bot-IoT dataset. In [22], Doshi et al. developed an attack detection algorithm comprised of feature extraction from the network traffic and ML classifier. In the place of the ML classifier, the authors used each of K-Nearest Neighbour (KNN), SVM, DT, and MLP methods; then, they evaluated the performance of this algorithm on a dataset collected within the same work. Letteri et al. [23] developed an MLP based Mirai Botnet detector specialized for Software Defined Networks. The authors fed 5 metrics, including the used communication protocol, to MLP.

In [24], Banerjee and Samantaray performed experimental work to deploy a network of honeypots that attracts botnet attacks and to detect those attacks via ML methods, such as DT, NB, Gradient Boosting, and RF. In reference [25], McDermott et al. developed the Bidirectional LSTM-based method which is developed for packet-level botnet attack detection by performing text recognition on multiple features including source and destination IP addresses of a packet. In addition, Tzagkarakis et al. [26] developed a sparse representation framework with parameter tuning using only normal traffic for botnet attack detection.

Meidan et al. [27] developed an ML-based attack detection technique which is trained using only normal traffic and tested for Mirai and Bashlite botnet attacks on an IoT network with nine devices. The authors also published the data collected in this study under the name N-BaIoT dataset. In order to detect Botnet attack in N-BaIoT dataset, Htwe et al. [28] used Classification and Regression Trees with feature selection, and Sriram et al. [29] performed a comparative study using 7 different ML methods (including NB, KNN, and SVM). In Reference [30], Soe et al. developed a Botnet attack detection algorithm comprised of two sequential phases first to train utilized ML method and perform feature selection, then to perform attack detection. The authors used MLP and NB within this architecture, and they evaluated the performance on N-BaIoT dataset. In [31], Parra et al. developed a cloud based attack detection method using Convolutional Neural Network (CNN) for phishing and using Long-Short Term Memory (LSTM) for Botnet attacks. The authors evaluated the performance of this method also on the N-BaIoT dataset achieving 94.8% accuracy. CNN was also used by Liu et al.

TABLE I
KEY FEATURES OF RELATED WORKS ON SELF-SUPERVISED LEARNING FOR INTRUSION DETECTION

Reference	IDS	Labeled Data	Data Generator	Learning Approach
Song and Kim [9]	Reduced Inception ResNet	Normal Traffic + Generated Noised Pseudo Normal Data	LSTM	Offline
Wang et al. [10]	BYOL Encoder + Linear Classifier	Malicious and Normal Traffic	BYOL	Offline (Transfer Learning)
Zhang et al. [11]	Deep Adversarial Anomaly Detection (DAAD)	Normal Traffic + Generated Feature Latents	GAN	Offline
Kye et al. [12]	AE-based Hierarchical Anomaly Detection	Only Normal Traffic	No Generator	Offline
Caville et al. [13]	GNN	None	GraphSAGE	Offline
Wang et al. [14]	AE-based Intrusion Score	None	Contextual Masking	Offline
Abououf et al. [15]	LSTM-AE	Only Normal Traffic	Auto Encoder-Decoder	Offline
Meyer et al. [16]	AE Neural Network	Reduced Data of Malicious and Normal Traffic	No Generator	Offline (Federated Learning)
SSID Framework	AADRNN	None	No Generator	Online

[32] with features that are processed by the triangle area maps based multivariate correlation analysis algorithm. In recent work [33], Bovenzi et al. employed DL models, specifically using Auto Encoders (AEs) and KitNET, for unsupervised early anomaly detection in IoT datasets, namely IoT-23 [34] and Kitsune [17]. The results of [33] demonstrated the potential for early anomaly detection in IoT network attacks by evaluating the detection effectiveness of varying numbers of packets, finding the first four packets to be the most effective.

B. Compromised Device Identification

Some recent work [35]–[40] focused on detecting compromised IoT devices during Botnet attacks, while Kumar et al. [35] detected Mirai-like bots scanning the destination port numbers in packet headers using an optimization-based technique for subsets of all IoT packets. Chatterjee et al. [36] identified malicious devices in IoT networks via evidence theory-based analysis. To this end, they analyzed traffic flows and selected the rarest features from a large number of communication features, including number of connections, transport layer protocol, and source/destination ports. In [41], in order to detect IoT botnet in an Industrial IoT network, Nguyen et al. developed a dynamic analysis technique utilizing various ML models, such as SVM, DT, and KNN, based on the features generated from the executable files. In Reference [42], Hristov and Trifonov developed a compromised device identification algorithm using wavelet transformation and Haar filter on the metrics indicating the processor, memory and network interface card usage of an IoT device. In [40], Prokofiev et al. used Logistic Regression to determine if the source device is a bot based on 10 metrics regarding the traffic packets. The performance of logistic regression is tested for a botnet that spreads through brute-force attacks. Nguyen et al. [37] detected compromised devices by an anomaly detection technique combining federated learning with language analysis for individual device types identified prior to detection. In

order to evaluate the performance of this technique, the authors collected a dataset by installing 33 IoT devices, 5 of which were malicious, and showed that detection performance is around 94% for positive and 99% for negative samples.

More differently, in Reference [38], Abhishek et al. detected not compromised devices but compromised gateways monitoring the downlink channels in an IoT network and performing binary hypothesis test. In [43], Trajanovski and Zhang developed a framework consisting of honeypots to identify the indicators of compromised devices and botnet attacks. Bahşi et al. in [44] addressed the scalability issues for ML-based Bot detection algorithms by minimizing the number of inputs of ML model via feature selection. In [39], for mobile IoT devices, Taneja proposed to detect compromised devices taking into account their location, such that if a location change or current location of an IoT device is classified as unusual behavior, the device is considered compromised.

C. Self-Supervised Learning for Intrusion Detection

Song and Kim [9] developed self-supervised learning algorithm for anomaly-based IDS in in-vehicle networks. In this algorithm, Reduced Inception-ResNet is used to make binary classification and detect unknown (zero-day) attacks. The training of this anomaly-based IDS offline uses both normal traffic and noised pseudo data generated using LSTM. Wang et al. [10] applied and adapted Bootstrap Your Own Latent (BYOL) self-supervised learning approach, which has been proposed in [45], for intrusion detection. The parameters of the BYOL algorithm are learned, then updated via transfer learning, using a dataset containing both normal and malicious traffic samples. For anomaly detection, Zhang et al. [11] developed deep adversarial training architecture by extending the well-known bidirectional Generative Adversarial Network (GAN) model. This architecture jointly learns from normal data and generated latent features. Kye et al. [12] introduced a hierarchical network IDS based on the Auto Encoder (AE).

By leveraging self-supervised signals and specialized anomaly scores within its AE architecture, this IDS learns offline only from normal traffic data and does not need an additional data generator. Caville et al. [13] developed a Graph Neural Network (GNN) based network-level IDS. This IDS was trained with a self-supervised learning approach using both positive and negative samples for offline training with an encoder graph created using an extended version of the well-known GraphSAGE framework. Wang et al. [14] developed an IDS with unsupervised learning combined with transformer based self-supervised masked context reconstruction, which improves the learning by magnifying the abnormal intrusion behaviours. Abououf et al. [15] developed a lightweight IDS architecture based on LSTM Auto Encoder (LSTM-AE) to perform detection on IoT nodes. This model is trained offline and unsupervised in an encoder-decoder architecture using a pre-collected dataset in the cloud. Meyer et al. [16] developed a federated self-supervised learning IDS. This IDS employ auto-encoder based self-supervised model on local datasets using federated approach. Note that the key features of the related works on self-supervised learning, as well as the proposed SSID framework, are summarized in Table I.

In this paper, we introduce a pioneering learning framework termed SSID, tailored for Deep Learning (DL)-based intrusion detection. Unlike conventional approaches prevalent in the ML / DL literature, SSID distinguishes itself in several key aspects:

- SSID eliminates dependency on additional generative models. Unlike common practice in self-supervised learning, which often necessitates the incorporation of an extra generative model (or contrastive method) for training purposes [46], our SSID framework operates without this requirement.
- SSID is independent of offline training data. While many methods in the literature rely on the availability of pre-collected (unlabeled) offline training data [47], SSID is designed to function autonomously, even in scenarios where such data is not readily accessible. However, if available, SSID can leverage offline data to enhance its performance.
- SSID facilitates fully online learning on independent network nodes. A distinctive feature of SSID is its ability to facilitate continuous online learning and enable real-time intrusion detection across independent network nodes.

III. INTRUSION DETECTION SYSTEM USED IN SSID

The SSID framework does not consider a specific algorithm for IDS or have strict requirements for it, except that it is based on ML / DL or some other function with learnable parameters and has a certain range of inputs and outputs. In addition, the SSID framework can also be used with both anomaly and signature based detection algorithms. On the other hand, as real-time network traffic contains only normal “benign” traffic until an attack occurs, an IDS structure that can learn only from normal traffic may provide higher performance under self-supervised learning. Therefore, anomaly-based algorithms are the main focus in this paper.

We first present the structure of anomaly-based IDS that we used within the SSID framework. This particular IDS structure

is displayed in Figure 1, which is mainly comprised of an DL model and a decision maker component.

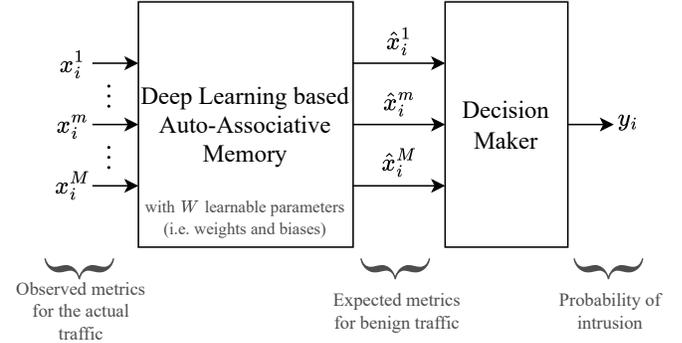


Fig. 1. Particular structure of IDS used within the SSID framework during performance evaluation

In the IDS structure shown in Figure 1, the DL model is used to create an Auto-Associative Memory (AAM) that reconstructs benign traffic metrics –which are the expected metrics according to the norm of the actual traffic learned by the AAM– from observed metrics. The significant difference between the reconstructed (expected) metrics and the actual metrics, as measured by the decision maker component, may be indicative of malicious traffic.

The input of this IDS is the vector of M network traffic metrics, $x_i = [x_i^1, \dots, x_i^m, \dots, x_i^M]$, and the output, y_i is a decision indicating the probability of intrusion corresponding to current traffic. The vector of expected metrics, which is the output of DL-based AAM, for packet i is denoted by $\hat{x}_i = [\hat{x}_i^1, \dots, \hat{x}_i^m, \dots, \hat{x}_i^M]$. The DL-based AAM is a learned function that maps the noisy or disordered metrics to the normal metrics, i.e. $f_{aam} : x_i \mapsto f_{aam}(x_i)$ for $f_{aam}(x_i) = \hat{x}_i$, so that $f_{aam} : [0, 1]^M \rightarrow [0, 1]^M$.

A. Deep Random Neural Network as Auto-Associative Memory

In the particular implementation of the IDS shown in Figure 1, in place of the DL algorithm, we use the DRNN model [48], [49]. DRNN is an extension of the RNN [50], [51] with dense feedback loops between clustered neuronal somas, and an overall feed-forward structure between layers of dense clusters. After it is trained to create an AAM, we call it Auto-Associative DRNN (AADRNN). Earlier research has shown that an AADRNN-based IDS has a lightweight architecture and offers high accuracy when used with unsupervised auto-associative training with normal (benign) traffic [7], [8], [52]–[55]. The AADRNN-based IDS was also evaluated with offline [54], incremental [7] and sequential [8] learning to detect malicious traffic and compromised devices during Botnet attacks. G-Networks [56], which generalize the RNN, and the simple RNN itself were also used with offline learning to detect zero-day [55] and SYN DoS [57] attacks. In contrast, in this paper, we use the AADRNN-based IDS as a specific IDS within the novel SSID framework proposed for the first time in this paper. The parameters of the AADRNN-based IDS are now learned online thanks to the SSID framework learning

approach that is completely online and does not require labeled data. It should be noted that the main contribution of this paper is the development of self-supervised approach and the novel SSID framework. In addition, AADRNN-based IDS can be replaced with another ML or DL based IDS, e.g. MLP will also be evaluated within the SSID framework in Section V.

In the AADRNN-based IDS, we use a DRNN model consists of H layers and a rectangular structure with equal units at each layer. Accordingly, we set $H = M$, each hidden layer $h \in \{1, \dots, H-1\}$ contains M neural clusters, and the output layer is comprised of M linear neurons.

B. Intrusion Detection Process

Recall that this paper considers two main tasks: detecting malicious traffic packets and identifying compromised network nodes. As the results of earlier research [8], [54] showed, each task requires a customized IDS to provide high accuracy. Therefore, in the remainder of this subsection, we present the traffic metrics and the decision maker component for each task separately.

1) *Traffic Metrics*: We use the traffic metrics that are defined in our earlier research specifically for Botnet attack detection in [54] and compromised device identification in [8]. These metrics are relatively few in number and are calculated based solely on packet header information. Therefore, they remain anonymous regarding packet content and communicating devices, do not need for any sensitive or device-specific information, thus preventing IDS from making biased decisions, and are suitable for real-time operation on lightweight systems. Meanwhile, they have been shown to be effective in capturing signatures of Botnet attacks.

For malicious traffic detection, in order to capture the signatures of Botnet attacks (especially Mirai), as the inputs of AAM, we use $M = 3$ normalized metrics that measure the total size and average inter-transmission times of the last 500 packets and the number of packets in the last 100 seconds.

For compromised device identification, we use $M = 6$ normalized traffic metrics which are calculated over time windows of length 10 seconds. These metrics measure the average size and average number of packets received from a single source, the maximum size and maximum number of packets received from any single source, and the total size and total number of packets transmitted during a time window.

2) *Estimation of the Expected Benign Traffic Metrics*: The expected benign traffic metrics is estimated (i.e. reconstructed) based on the traffic metrics given as the inputs of AADRNN. The estimated traffic metrics (\hat{x}_i) are then fed into the decision maker component.

Let W_h denote the $[(M+1) \times M]$ matrix of connection weights (including biases) between the layer $h-1$ and layer h for $h \in \{1, \dots, H\}$; that is, W_h is the multiplier for the inputs of layer h . In addition, $\zeta(\cdot)$ denote the activation function of a cluster in AADRNN.

Accordingly, in real-time operation, the forward pass of AADRNN model for the given input vector x_i is computed

as:

$$\hat{x}_{(i,1)} = \zeta([x_i, 1] W_1) \quad (1)$$

$$\hat{x}_{(i,h)} = \zeta([\hat{x}_{(i,h-1)}, 1] W_h) \quad \forall h \in \{2, \dots, H-1\}, \quad (2)$$

$$\hat{x}_i = [\hat{x}_{(i,H-1)}, 1] W_H, \quad (3)$$

where $\hat{x}_{(i,h)}$ is the output of layer h for packet i , and the term $[x_i, 1]$ or $[\hat{x}_{(i,h)}, 1]$ indicates that 1 is added to the input of each layer as a multiplier of the bias.

The activation function $\zeta(\cdot)$ is defined as [48]:

$$\zeta(\Lambda) = \frac{p(r + \lambda^+) + \lambda^- + \Lambda}{2[\lambda^- + \Lambda]} \quad (4)$$

$$- \sqrt{\left(\frac{p(r + \lambda^+) + \lambda^- + \Lambda}{2[\lambda^- + \Lambda]}\right)^2 - \frac{\lambda^+}{\lambda^- + \Lambda}},$$

where Λ is the input of the given cluster, p is the probability that any neuron received trigger transmits a trigger to some other neuron, and λ^+ and λ^- are respectively the rates of external Poisson flows of excitatory and inhibitory input spikes to any neuron.

3) *Decision Making*: The last step in the attack detection process is making the attack decision. Since the AADRNN output provides the expected metrics for normal network traffic, any deviation of the actual metrics from the AADRNN output, i.e. the expected metrics, indicates malicious traffic. Therefore, we use a lightweight decision making approach based on comparison of actual and expected metric values. In particular, we measure the deviation of the actual metrics x_i from the expected metrics \hat{x}_i . On the other hand, since AADRNN-based IDS uses different set of metrics for malicious traffic detection and compromised device identification tasks, the decision making approach is also slightly different for each task. One should note that this lightweight decision making approach requires no human intervention or parameter settings based on offline data; therefore, it is suitable for online learning IDS.

For malicious traffic detection, the decision maker calculates the output of IDS, y_i , as the average absolute difference between the actual and the expected metric values:

$$y_i = \frac{1}{M} \sum_{m=1}^M |x_i^m - \hat{x}_i^m|. \quad (5)$$

For compromised device identification, the decision maker calculates the output of IDS as the maximum absolute difference between the actual and the expected metric values:

$$y_i = \max_{m \in \{1, \dots, M\}} (|x_i^m - \hat{x}_i^m|). \quad (6)$$

C. Learning Algorithm to Create AADRNN

In order to create an AADRNN, we train an DRNN model based on a batch of packet samples, denoted by B^l , collected and provided by the proposed SSID framework at any learning phase l . To this end, we use the learning algorithm, used for DRNN in earlier research [7], [8], [54], which shall be reviewed in this subsection.

1) *Initial Learning*: As we use sequential learning to create an auto-associative memory from DRNN (namely, AADRNN), each weight matrix W_h is learned and updated based on only the batch of benign packets B^l provided by the SSID framework.

When the learning algorithm is called by SSID for B^l at initial learning phase $l = 0$, we solve the following problem using A Fast Iterative Shrinkage-Thresholding Algorithm (FISTA) [58]:

$$W_h = \underset{\{W: W \geq 0\}}{\operatorname{argmin}} \left(\left\| \left[\operatorname{adj}(\zeta(\hat{X}_{l,h-1}^{\operatorname{train}} W_R)), \mathbf{1}_{k \times 1} \right] W - \hat{X}_{l,h-1}^{\operatorname{train}} \right\|_{L_2}^2 + \|W\|_{L_1} \right), \quad (7)$$

where W_R is a random weight matrix whose elements are in $(0, 1)$ and $\hat{X}_{l,h}^{\operatorname{train}}$ is the matrix that collects the output vectors of layer h for all packets $i \in B^l$ if $y_i = 0$:

$$\hat{X}_{l,h}^{\operatorname{train}} = \{\hat{x}_{(i,h)}, \forall i \in B^l\}, \quad (8)$$

$$\hat{X}_{l,0}^{\operatorname{train}} = \{x_i, \forall i \in B^l\}, \quad \text{and} \quad \hat{X}_{l,H}^{\operatorname{train}} = \{\hat{x}_i, \forall i \in B^l\}$$

After FISTA is performed for a predefined number of iterations to solve (7), we normalize the resulting weight matrix W_h :

$$W_h \leftarrow 0.1 \frac{W_h}{\max(\hat{X}_{i,h}^{\operatorname{train}})}. \quad (9)$$

2) *Online Incremental Learning*: During the online incremental learning of IDS parameters (i.e. $l \geq 1$), only the connection weights of the output layer of AADRNN, W_H^l , are updated via the second stage to learn the new patterns of the benign traffic provided by SSID via B^l . As the incremental learning algorithm, we mainly integrate the sequential learning algorithm developed in [59]. Let define the operation matrix O_l , which is initialized for $l = 0$ as the inverse of the Gram matrix:

$$O_0 = \left[(\hat{X}_{l,H}^{\operatorname{train}})^T \hat{X}_{l,H}^{\operatorname{train}} \right]^{-1} \quad (10)$$

If there is at least one benign packet in B^l , we first compute the value of O_l for $l \geq 1$:

$$O_l = O_{l-1} - O_{l-1} (\hat{X}_{l,H-1}^{\operatorname{train}})^T \left[I + \hat{X}_{l,H-1}^{\operatorname{train}} O_{l-1} (\hat{X}_{l,H-1}^{\operatorname{train}})^T \right]^{-1} \hat{X}_{l,H-1}^{\operatorname{train}} O_{l-1} \quad (11)$$

Then, the value of W_H is updated:

$$W_H \leftarrow W_H + O_l (\hat{X}_{l,H-1}^{\operatorname{train}})^T (\hat{X}_{l,0}^{\operatorname{train}} - \hat{X}_{l,H-1}^{\operatorname{train}} W_H) \quad (12)$$

3) *Learning Error Provided to the SSID Framework*: Furthermore, since we use an anomaly-based algorithm that learns only the benign traffic, we take the learning error as the mean of estimated attack probabilities for packets in learning batch B^l :

$$\mathcal{E}(l) = \frac{1}{|B^l|} \sum_{i \in B^l} y_i \quad (13)$$

IV. THE SELF-SUPERVISED INTRUSION DETECTION FRAMEWORK

As the main contribution of this paper, we propose the novel SSID framework to enable fully online self-supervised learning of the parameters of IDS with no need for human intervention. In order to clearly present the SSID framework, this section first explains the main functionalities of both the initial and online learning phases through Figure 2. Then, the comprehensive methodology, that includes the details regarding all the blocks in Figure 3, is presented.

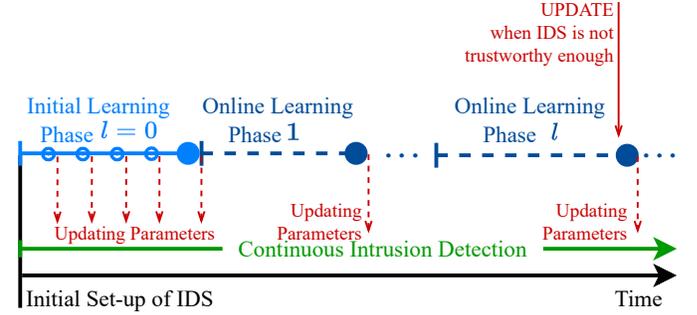


Fig. 2. Detection and learning processes of IDS within the Fully Online Self-Supervised Intrusion Detection (SSID) framework

As shown in Figure 2, within the SSID framework, there are two main operations performed, intrusion detection (lower line) and learning (upper line). Intrusion detection is the main operation performed by IDS and is not modified by SSID. That is, intrusion detection (as an operation) is defined only by a particular IDS algorithm used in SSID. On the other hand, we can say that our SSID framework ensures that IDS makes accurate decisions by updating its parameters with online self-supervised learning, and it performs intrusion detection uninterruptedly and continuously. Regarding the communication (data transfer) between intrusion detection and online self-supervised learning processes in SSID, it is important to note that SSID uses the decisions of IDS to enable self-supervised learning during either initial or online learning phases.

A. Online Self-Supervised Learning

In parallel with attack detection, our SSID framework provides online self-supervised learning of IDS parameters, as shown on the upper process line in Figure 2. As seen in that figure and Figure 3 which shows the learning process in SSID, the online self-supervised learning process starts with the initial learning phase (namely, $l = 0$) and continues with successive online learning phases.

Since network traffic characteristics may vary with time and substantially affect the IDS detection performance, it is crucial to update the parameters of the IDS online concurrently with its real-time operation. The parameter updates conducted through online learning serve to enhance the reliability of the IDS as a detector by enhancing its performance and ensuring it remains aligned with the latest traffic characteristics. Online learning also circumvents the need to collect and label extensive datasets for offline training, thereby conserving both time

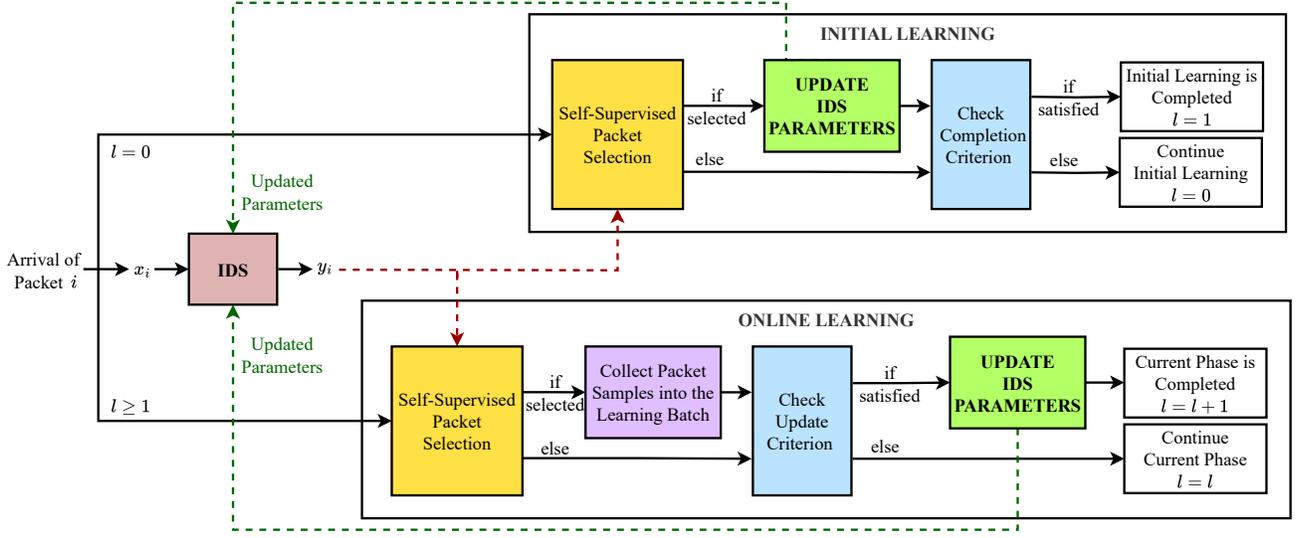


Fig. 3. Block diagram of the learning process in the SSID framework for online self-supervised learning of the parameters of IDS

and resources. In situations where labeled datasets are readily accessible, the IDS can be pre-trained, and its performance can be validated using those datasets.

1) *Initial Learning*: The initial learning phase in SSID can be considered a special case of the proposed methodology of self-supervised learning, which allows IDS to be used from its initial setup and updates the parameters of the IDS frequently achieving the desired performance gradually and quickly.

In detail, as shown in the top block of Figure 3, during the initial learning phase in SSID, the parameters of the IDS are updated for each packet that is selected for learning via our self-supervised packet selection methodology. Whether the parameters of the IDS are updated or not, SSID checks the trust based completion criterion of the initial learning phase $l = 0$ aiming to complete this phase as soon as the IDS is trained enough to make trustworthy decisions. To this end, it first calculates the trustworthiness of the IDS, namely the “trust coefficient” denoted by $\Gamma \in [0, 1]$, which indicates the confidence of SSID in any decision made by the IDS. Since the IDS does not have any information about the network traffic patterns yet, SSID cannot judge the decisions of the IDS and starts the initial learning process with $\Gamma = 0$ meaning that there is no trust in the decisions of the IDS.

Subsequently, SSID checks the trust criterion to complete the initial learning phase $l = 0$ by measuring if SSID’s trust in the IDS is greater than a threshold Θ , is the minimum desired trust level:

$$\text{if } \Gamma \geq \Theta, \text{ complete initial learning and set } l = 1 \quad (14)$$

Thus, as given in (14), if $\Gamma \geq \Theta$, the initial learning phase has been completed, and the next packet will be considered for the first phase $l = 1$ of continuous online learning.

2) *Online Learning*: After the initial learning is completed, the parameters of IDS are updated via an online learning phase $l \geq 1$ when the trust of SSID in the IDS is unacceptably low. As the lower block in Figure 3 shows, the parameters of the

IDS are updated for a collected batch of packets when the trustworthiness of the IDS is not acceptable anymore. When SSID is in the online learning phase $l \geq 1$, each packet i selected by our self-supervised packet selection method is collected into the batch of training packets, B^l .

Then, SSID checks the trust-based criterion to update the parameters of the IDS. Inversely with the initial learning phase, SSID now updates the parameters of the IDS if $\Gamma < \Theta$, at least K packets are collected for learning (i.e. $|B^l| \geq K$), and there is no attack detected by the IDS:

$$\text{if } \Gamma < \Theta \text{ and } |B^l| \geq K \text{ and } \frac{1}{I} \sum_{j=i-I+1}^i y_j \leq \gamma, \\ \text{update parameters and set } l = l + 1 \quad (15)$$

where I is the number of packets to calculate the average of the intrusion decisions, γ is the intrusion threshold, and K is provided by the user considering properties of the network and learning algorithm. Limit of minimum K packets is added only to provide practical efficiency for training.

That is, by (15), SSID waits for a considerable decrease in the trustworthiness of the decisions of IDS to update the parameters since Γ is known to be already greater than Θ at the end of the initial learning phase $l = 0$. In this way, the learning is performed when it is essential.

On the other hand, if an intrusion is detected where the average output of the IDS is greater than γ , SSID clears the batch of collected packet samples, B^l :

$$\text{if } \frac{1}{I} \sum_{j=i-I}^i y_j > \gamma, \text{ empty } B^l \quad (16)$$

With this cleanup, SSID aims to prevent the IDS from learning any false negative instances since false negative outputs are very likely to occur just before an attack is detected.

B. Self-Supervised Packet Selection

In the remainder of this section, we present our proposed methodology to train the utilized IDS in a self-supervised fashion enabling the fully online property of SSID. In other words, we explain the details of the learning process in SSID, which are shown as subblocks in Figure 3.

As the first operation of the learning process in SSID, each packet i is decided to be used in learning for the next update of IDS parameters. The packet selection is executed in a self-supervised manner that only considers the output of the IDS together with SSID's trust in it.

Let p_i^- and p_i^+ respectively be the probability of selecting packet i to be used as a *benign* or *malicious* packet sample in the training of IDS, and q_i be the probability of rejecting i to be used in training. That is, we select the packet i as the sample of a benign packet with probability p_i^- or that of an attack packet with probability p_i^+ to use it in training, or the packet i is not included in the training set with probability q_i . Also, recall that $y_i \in [0, 1]$ is the output of IDS for packet i .

Since we assume that there are no packet labeling mechanisms or labor to prepare packet data for learning, we select each packet i based on the output of IDS (which is the estimation of the probability of packet i being malicious) considering how trustworthy IDS is. Therefore, we shall also define a trust coefficient Γ to measure the trustworthiness of IDS at any time based on the representativeness of the packet samples that IDS learned until the end of the last learning phase and the generalization ability of IDS from these samples.

Accordingly, we start by defining p_i^+ as

$p_i^+ \equiv$ (trust in IDS) (est. prob. of packet i being malicious)

$$p_i^+ = \Gamma y_i \quad (17)$$

We further define p_i^- similarly to p_i^+ :

$p_i^- \equiv$ (trust in IDS) (est. prob. of packet i being normal)

$$p_i^- = \Gamma(1 - y_i) \quad (18)$$

Subsequently, since

$$p_i^+ + p_i^- + q_i = 1, \quad (19)$$

the probability q_i of not selecting the packet i for training is:

$$\begin{aligned} q_i &= 1 - (p_i^+ + p_i^-) \\ &= 1 - \Gamma \end{aligned} \quad (20)$$

Recall that SSID starts with $\Gamma = 0$ since the IDS does not have yet any information about the network traffic patterns at the initial learning phase. That is, the output of the IDS is calculated using the initial parameter values (if available) and will not be able to achieve accurate detection for the particular traffic. In addition, for selecting the first packet, the parameters of the IDS are updated for the first time using $p_i^- = 1$, $p_i^+ = 0$, and $q_i = 0$. Thus, SSID selects the first packet to learn as a benign sample.

C. Trustworthiness of IDS

Now, we determine the trust coefficient Γ for the IDS in the SSID framework. Through this coefficient, we aim to include both the effects of changes in the normal behavior of network traffic over time and the generalization ability of the IDS into the packet selection model for learning.

To this end, we first define the factor of ‘‘representativeness’’, denoted by C_{rep} , for the traffic packets that are learned by the IDS. The representativeness factor C_{rep} takes a value in the range of $[0, 1]$ and measures how much the packets used for learning (during all of the past learning phases) represent the total observed traffic.

In addition, we define the factor of ‘‘generalization ability’’, denoted by C_{gen} , of the IDS. The generalization factor C_{gen} takes a value in the range of $[0, 1]$ and is calculated only at the end of each parameter update since it is the only time when the parameters of the IDS are updated. These two factors shall respectively be given in Section IV-D and Section IV-E.

Accordingly, in (21), we determine the trust coefficient Γ as the multiplication of C_{rep} and C_{gen} :

$$\Gamma = C_{rep} C_{gen} \quad (21)$$

In this way, Γ simultaneously measures how much the IDS is able to learn and generalize from provided traffic packets and how much these packets reflect actual traffic patterns. That is, through this trust coefficient, we evaluate how much information the IDS can generalize from the traffic packets provided to make decisions for the upcoming traffic.

D. Representativeness of the Traffic that is Learned (C_{rep})

In order to calculate the representativeness of the packet traffic used during the earlier learning phases, we compare the learned traffic with the total observed traffic through Kullback-Leibler (KL)-Divergence [60]. Therefore, there are two sets of traffic packets for comparison, the packets used in the previous learning phases up to and including l (where l is the latest completed learning phase) and the normal packets that are observed by IDS during continuous detection.

During this comparison, we assume that the packet traffic consists of two main properties, inter-transmission time (TT) and the packet length (PL) since these properties can be considered as the basis of traffic metrics, which are the inputs of the IDS. Inter-transmission time (TT) refers to the duration between the transmissions of consecutive packets, while packet length (PL) denotes the size of each packet in the network traffic. These properties are crucial as they directly reflect the behaviour and characteristics of network traffic, thus serving as fundamental metrics for IDS to analyse and detect anomalies or malicious activities effectively.

We further assume that packet arrivals –any sample collected from the network traffic– has a Poisson distribution so that the inter-transmission time TT is an Exponentially-distributed random variable. The packet length PL is also assumed to be an Exponentially-distributed random variable because the header length is considerably larger than the message length for the majority of IoT applications. In addition, TT and PL are considered to be independent. On the other

hand, for particular applications, these assumptions and the traffic model can be changed and the below methodology can easily be adapted for the new traffic model with a new set of assumptions.

Furthermore, let S_l^{TT} and S_l^{PL} respectively denote the sets of the inter-transmission times and lengths of all packets learned at the end of l , and S_o^{TT} and S_o^{PL} respectively denote the same of all normal packets observed during continuous detection. In addition, according to our assumptions, S_l^{TT} and S_o^{TT} have exponential distributions with means of $1/\lambda_l$ and $1/\lambda_o$ while S_l^{PL} and S_o^{PL} also have exponential distributions with means of $1/\mu_l$ and $1/\mu_o$.

1) *KL-Divergence for Inter-Transmission Times*: For the set of inter-transmission times, $D_{KL}(S_o^{TT}||S_l^{TT})$ is KL-Divergence from S_l^{TT} to S_o^{TT} measuring the information gain achieved if S_o^{TT} would be used instead of S_l^{TT} which has been used during the learning phases of SSID. Note that small KL-Divergence means low information gain, and $D_{KL}(S_o^{TT}||S_l^{TT}) = 0$ shows that S_o^{TT} and S_l^{TT} provide the same amount of information. Accordingly, using the definition of KL-Divergence [60], we first calculate $D_{KL}(S_o^{TT}||S_l^{TT})$, which can shortly be denoted by D_{KL}^{TT} , as

$$\begin{aligned} D_{KL}^{TT} &= \int_{-\infty}^{\infty} f(x; \lambda_o) \log\left(\frac{f(x; \lambda_o)}{f(x; \lambda_l)}\right) dx \quad (22) \\ &= \mathbb{E}_{f(x; \lambda_o)} \left[\log\left(\frac{f(x; \lambda_o)}{f(x; \lambda_l)}\right) \right] \\ &= \mathbb{E}_{f(x; \lambda_o)} \left[\log\left(\frac{\lambda_o}{\lambda_l}\right) - x(\lambda_o - \lambda_l) \right] \end{aligned}$$

where $f(x; \lambda_o)$ and $f(x; \lambda_l)$ denote the probability distribution functions of S_o^{TT} and S_l^{TT} respectively with parameters λ_o and λ_l . This leads to the result of

$$D_{KL}^{TT} = \log\left(\frac{\lambda_o}{\lambda_l}\right) - \frac{(\lambda_o - \lambda_l)}{\lambda_o} \quad (23)$$

2) *KL-Divergence for Packet Lengths*: Similarly with transmission times, for the set of packet lengths, $D_{KL}(S_o^{PL}||S_l^{PL})$ is KL-Divergence from S_l^{PL} to S_o^{PL} , which is shortly denoted by D_{KL}^{PL} , and is calculated as

$$\begin{aligned} D_{KL}^{PL} &= \int_{-\infty}^{\infty} f(x; \mu_o) \log\left(\frac{f(x; \mu_o)}{f(x; \mu_l)}\right) dx \quad (24) \\ &= \mathbb{E}_{f(x; \mu_o)} \left[\log\left(\frac{f(x; \mu_o)}{f(x; \mu_l)}\right) \right] \\ &= \mathbb{E}_{f(x; \mu_o)} \left[\log\left(\frac{\mu_o}{\mu_l}\right) - x(\mu_o - \mu_l) \right] \end{aligned}$$

where $f(x; \mu_o)$ and $f(x; \mu_l)$ denote the probability distribution functions of S_o^{PL} and S_l^{PL} respectively with parameters μ_o and μ_l . This results in:

$$D_{KL}^{PL} = \log\left(\frac{\mu_o}{\mu_l}\right) - \frac{(\mu_o - \mu_l)}{\mu_o} \quad (25)$$

3) *Representativeness Factor based on Normalized KL-Divergence*: For both transmission times and packet lengths, we now obtained the KL-Divergence between the set of observed packets and the set of packets learned. However, the KL-Divergence cannot directly be used as a representativeness factor because of the following reasons: 1) It has no upper

bound but the representativeness factor $C_{rep} \in [0, 1]$. 2) KL-Divergence is decreasing function of the similarity between two sets but we need an increasing function of that as the name ‘‘representativeness’’ suggests. 3) This factor should be the combination of D_{KL}^{TT} and D_{KL}^{PL} .

Therefore, in order to obtain the representativeness factor, we first normalize each of D_{KL}^{TT} and D_{KL}^{PL} as

$$D_{KL-norm}^{TT} = e^{-D_{KL}^{TT}}, \quad (26)$$

$$D_{KL-norm}^{PL} = e^{-D_{KL}^{PL}}. \quad (27)$$

which solve the issues 1) and 2) stated above. Each of these normalized divergence measures can also be written in terms of only the traffic parameters:

$$\begin{aligned} D_{KL-norm}^{PL} &= e^{-\left[\log\left(\frac{\lambda_o}{\lambda_l}\right) - \frac{(\lambda_o - \lambda_l)}{\lambda_o}\right]} \\ &= \left[\frac{\lambda_l}{\lambda_o} e^{-\frac{(\lambda_l - \lambda_o)}{\lambda_o}} \right] \end{aligned} \quad (28)$$

Similarly,

$$\begin{aligned} D_{KL-norm}^{PL} &= e^{-\left[\log\left(\frac{\mu_o}{\mu_l}\right) - \frac{(\mu_o - \mu_l)}{\mu_o}\right]} \\ &= \left[\frac{\mu_l}{\mu_o} e^{-\frac{(\mu_l - \mu_o)}{\mu_o}} \right] \end{aligned} \quad (29)$$

Then, we combine $D_{KL-norm}^{TT}$ and $D_{KL-norm}^{PL}$ into the ‘‘representativeness factor’’ C_{rep} as

$$C_{rep} = c_1 D_{KL-norm}^{TT} + c_2 D_{KL-norm}^{PL} \quad (30)$$

where $c_1 \leq 1$ and $c_2 \leq 1$ are positive constants that satisfy $c_1 + c_2 = 1$.

In order to weigh transmission times and packet lengths equally, we take $c_1 = c_2 = 0.5$. That is, we take their average:

$$\begin{aligned} C_{rep} &= \frac{1}{2} [D_{KL-norm}^{TT} + D_{KL-norm}^{PL}] \quad (31) \\ &= \frac{1}{2} [e^{-D_{KL}^{TT}} + e^{-D_{KL}^{PL}}] \end{aligned}$$

We can rewrite (31) only in terms of the traffic parameters using (28) and (29):

$$C_{rep} = \frac{1}{2} \left[\frac{\lambda_l}{\lambda_o} e^{-\frac{(\lambda_l - \lambda_o)}{\lambda_o}} + \frac{\mu_l}{\mu_o} e^{-\frac{(\mu_l - \mu_o)}{\mu_o}} \right] \quad (32)$$

E. Generalization Ability of IDS (C_{gen})

As stated above, we consider the generalization ability of the IDS as one of two factors that define the trustworthiness of intrusion decisions. To this end, the aim of this subsection is to determine the generalization ability of the IDS in simple terms to make its computation as easy as possible using the available measures during the execution of SSID. Accordingly, we start with the basic definition of generalization [61]:

$$\text{Generalization} \equiv \text{Data} + \text{Knowledge}$$

stating that the generalization depends on the ‘‘Data’’, which is denoted by Δ and refers to the adequacy of the packet samples that are used for learning, and the ‘‘Knowledge’’, which is denoted by κ and refers to the knowledge of the IDS obtained

from packets learned. Therefore, we define the generalization factor C_{gen} as

$$C_{gen} = c_3 \Delta + c_4 \kappa \quad (33)$$

where c_3 and c_4 are positive constants such that $c_3, c_4 \leq 1$, and $c_3 + c_4 = 1$.

1) *Data Adequacy* (Δ): We evaluate the adequacy of the packet samples that are used for learning with respect to the number of learnable parameters in the IDS. Although there is no hard rule for determining the adequacy of the learning data (i.e., the number of training samples required) for a given DL model, most studies have shown its relationship to the total number of learnable parameters in the model and taken the minimum number of required training samples as a multiple of the number of parameters [62].

Therefore, we define the Δ as the counterpart of the ratio of the number of learnable parameters in the IDS to the total number of packet samples used for learning up to and including learning phase l :

$$\Delta = \left[1 - \min\left(\frac{W}{\sum_{k=0}^l |B^k|}, 1\right) \right] \quad (34)$$

where $\sum_{k=0}^l |B^k|$ is the total number of packet samples that are sequentially used to learn model parameters until the end of learning phase l . Clearly, Δ takes value in $[0, 1]$. While W is a constant number and $|B^l| \geq 1$ for any learning phase l (in which a learning is performed), $\lim_{l \rightarrow \infty} (\Delta) = 1$. In addition, $\Delta = 0$ when $\sum_{k=0}^l |B^k| \leq W$.

2) *Knowledge* (κ): We consider knowledge to be the measure of the DL model's expected performance for the upcoming traffic packets. Subsequently, we measure the knowledge (i.e. expected performance) of the IDS based on its performance on the packet samples used for learning and on the online available validation data.

To this end, in this paper, we consider the worst-case scenario when there is no validation data available. $\mathcal{E}(l)$ is the empirical error measured at the end of learning phase l on both packet samples learned and validation data (if available), such that $0 \leq \mathcal{E}(l) \leq 1$. We then define the knowledge κ as the counterpart of the exponentially weighted moving average of empirical errors for all learning phases up to and including the l -th phase:

$$\kappa = 1 - \sum_{k=0}^l \left(\frac{1}{2}\right)^{(l-k+1)} \mathcal{E}(k) \quad (35)$$

where the multiplier is set as $1/2$ to keep the value of κ in $[0, 1]$. That is, if the empirical training error decreases with the successive learning phases (i.e. $\mathcal{E}(l)$ is the decreasing function of l), the knowledge of the IDS increases converging to its maximum.

In practice, at the end of each learning phase l , κ can easily be updated using only its previous value and the empirical error $\mathcal{E}(l)$ as

$$\kappa \leftarrow \frac{1}{2} - \frac{1}{2} \left[\mathcal{E}(l) - \kappa \right] \quad (36)$$

3) *Generalization Factor*: We now easily calculate the generalization factor C_{gen} combining the ‘‘data adequacy’’ Δ (34) and ‘‘knowledge’’ κ (35) using the definition of the generalization factor (33):

$$C_{gen} = c_3 \left[1 - \min\left(\frac{W}{\sum_{k=0}^l |B^k|}, 1\right) \right] + c_4 \left[1 - \sum_{k=0}^l \left(\frac{1}{2}\right)^{(l-k+1)} \mathcal{E}(k) \right] \quad (37)$$

We particularly set $c_3 = c_4 = 0.5$ representing that the data and knowledge are equally important for generalization:

$$C_{gen} = \frac{\min(W/\sum_{k=0}^l |B^k|, 1) + \sum_{k=0}^l (1/2)^{(l-k+1)} \mathcal{E}(k)}{2} \quad (38)$$

V. RESULTS

We now evaluate the performance of SSID framework for two different intrusion detection tasks to identify malicious traffic packets and compromised devices.

A. Parameter Settings for SSID

First, we set the parameters of SSID as follows: trust threshold $\Theta = 0.95$, number of packets observed for decision $I = 10$, minimum number of training packets $K = 100$, and intrusion threshold $\gamma = 0.25$. That is, SSID aims to keep the trust in the IDS above 0.95 while it considers a packet as malicious if the output of the IDS is above 0.25. IDS analyses $I = 10$ packets to make robust intrusion decisions, which allows IDS to make early decisions while not being too reactive to instantaneous changes. In addition, the parameters are updated using at least $K = 100$ packets in a learning batch for computational efficiency.

B. Datasets

Since the proposed SSID framework provides online learning for the IDS, its performance is evaluated using two well-known datasets, **Kitsune** [17], [63] and **Bot-IoT** [18], which contain the actual packet transmissions for both normal and malicious traffic over time. These are two of the most recent and used datasets on Botnet and DDoS attacks.

Some other examples of such datasets are UNSW-NB15 [64], CICIDS 2017 [65], and IoT-23 [34]. The UNSW-NB15 dataset focuses on general network intrusion detection, not specifically focuses on botnet activities, capturing a wide range of network activities, while CICIDS 2017 emphasizes both traditional and IoT-specific attacks in controlled lab environments. IoT-23 is tailored for studying IoT-specific security challenges, offering data from various IoT devices. Meanwhile, Bot-IoT and Kitsune datasets specifically focuses on IoT DoS and DDoS attacks. They include realistic IoT device behaviors and various types of botnet activities, such as command and control communication, malware propagation, and reconnaissance scans. These datasets offer a comprehensive collection of normal and malicious traffic data for benchmarking attack detection and compromised device identification systems.

In the rest of this section, for malicious traffic detection and compromised device identification during Mirai Botnet attack, we first use the well-known Kitsune dataset [17], which contains 764,137 packet transmissions of both normal and attack traffic cover a consecutive time period of roughly 7137 seconds. Of the total 764,137 packets exchanged between 107 distinct IP addresses in this dataset, 121,621 are normal packets and 642,516 are malicious packets. Then, for compromised device identification, in addition to the Mirai Botnet, we use the following data: 1) SYN DoS attack from the Kitsune dataset, 2) DDoS attacks using HTTP, TCP and UDP protocols from the Bot-IoT dataset, and 3) DoS attack using HTTP protocol from the Bot-IoT dataset. The data of SYN DoS, DDoS HTTP, DDoS TCP, DDoS UDP, and DoS HTTP are respectively comprised of “2,771,276”, “19,826”, “19,548,235”, “18,965,736”, and “29,762” packets.

C. Performance Evaluation for Malicious Traffic Detection

The performance of SSID is first evaluated for malicious traffic detection during Mirai Botnet attack. Figure 4 displays the ROC curve, where the x-axis of this figure is plotted in logarithmic scale. We see that AADRNN-based IDS trained under our novel SSID framework achieves significantly high TPR above 0.995 even for very low FPR about 10^{-5} .

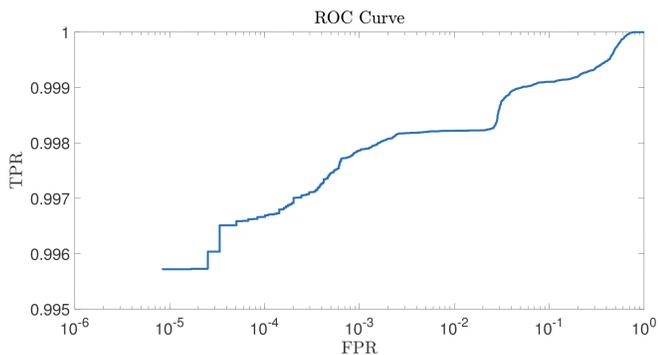


Fig. 4. ROC curve for the performance of AADRNN-based IDS under the SSID framework for malicious traffic detection

In more detail, in Figure 5, we present the predictions and Γ of SSID with respect to time. This figure reveals an important fact that while the IDS is completely indecisive at the beginning, SSID framework enables it to learn the normal traffic very quickly. As a result, SSID makes significantly low false alarms although it learns –fully online– during real-time operation based only on its own decision using no external (offline collected) dataset. We also see that Γ accurately reflects the trustworthiness of decisions made by AADRNN. In addition, although Γ slightly decreases as a result of random packet selection, especially after attack starts, the parameters of AADRNN are not updated by SSID as the traffic is detected as malicious.

1) Comparison with Incremental and Offline Learning:

We further compare the performance of AADRNN under SSID with the performance of AADRNN with incremental and offline learning. All methods with offline learning are trained using the first 83,000 benign traffic packets while the

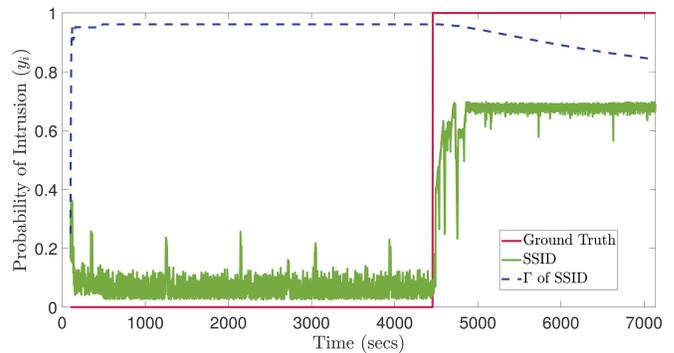


Fig. 5. Predictions of SSID and the value of trust coefficient Γ with respect to time

AADRNN with incremental learning is trained periodically for the window of 750 packets using AADRNN’s own decision, where the first 750 packets received are assumed to be normal packets during the cold-start of the network.

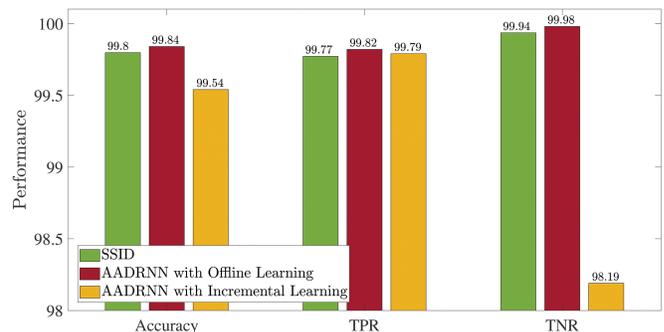


Fig. 6. Performance comparison between the AADRNN under SSID and the AADRNN with incremental and offline learning

Figure 6 displays the performances of SSID and the AADRNN, with incremental and offline learning. The results in this figure first reveal that the fully online trained AADRNN using the SSID framework achieves competitive results with the AADRNN which is trained offline using approximately 83,000 packets. We also see that the SSID significantly outperforms the AADRNN with incremental learning with respect to all performance metrics. Also note that the SSID learned from a total of 4,161 packets while also conducting real-time attack detection.

In contrast with offline and incremental learning, SSID framework assumes only that the first packet is known to be benign so the duration of cold-start equals the transmission of a single traffic packet. That is, using no offline dataset or requiring no cold-start, the SSID framework is able train an DL-based IDS to achieve considerably high performance which is highly competitive against the DL models trained on significantly large dataset.

2) A Different DL Model –MLP– under the SSID Framework: During the performance evaluation of the SSID framework, we also use MLP, which is one of the most popular feed-forward neural networks used for various tasks such as

signal processing, forecasting, anomaly detection, etc. As also reviewed in Section II, various works [19], [22], [23], [30] used MLP to develop different IDS methods.

Similar to the AADRNN, the MLP model that we use is also comprised of M layers with M neurons each. Each neuron has *sigmoid* activation function as

$$\zeta(\Lambda) = \frac{1}{1 + e^{-\Lambda}}, \quad (39)$$

where Λ is an input to the neural activation.

In both the initial and online learning stages, the parameters of the MLP are updated using the state-of-the-art optimizer Adam. In each online learning phase, incremental learning is applied by starting parameter optimization from the connection weight values already in use at the beginning of that phase.

In order to further analyze the impact of the proposed SSID framework on the performance of a different DL model, we evaluate the performance of the well-known MLP under the SSID framework (called SSID-MLP) and compare it with the performance of MLP with offline and incremental learning, respectively. The results of this performance evaluation is presented in Figure 7.

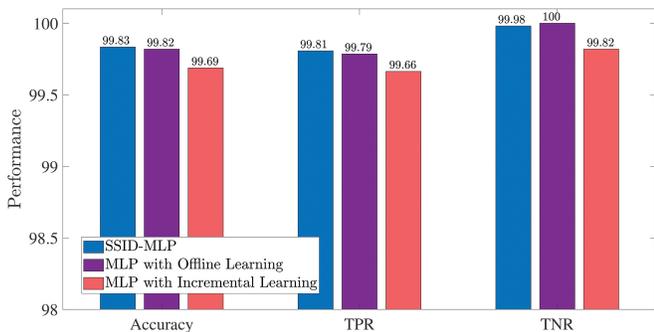


Fig. 7. Performance comparison between the SSID-MLP and the MLP with incremental and offline learning

Figure 7 shows that SSID-MLP achieves slightly higher Accuracy and TPR than MLP with offline learning, although MLP with offline learning raises no false alarms at all (i.e. with 100% TNR). Moreover, we see that SSID-MLP significantly outperforms the MLP model that is also trained via incremental learning periodically for every 750 packets based on its own output.

3) *Comparison of Different DL Models*: We further compare the performance of AADRNN under SSID (called SSID-AADRNN for clarity) and SSID-MLP with those of some well-known ML models, including KNN and Lasso with offline learning. Note that models with offline learning are trained using a labeled normal traffic data. Also, note that Lasso, KNN, and MLP are well-known and commonly used models for Botnet attack detection and compromised device identification [22], [23], [29], [41]. Figure 8 displays the performance of all compared models with respect to Accuracy, TPR and TNR. The results in this figure show that SSID-MLP achieves the second-best performance with respect to all performance metrics. In addition, both SSID-MLP and SSID-AADRNN achieve highly competitive results with the offline

trained DL models, while the SSID framework completely eliminates the need for data collection and labeling.

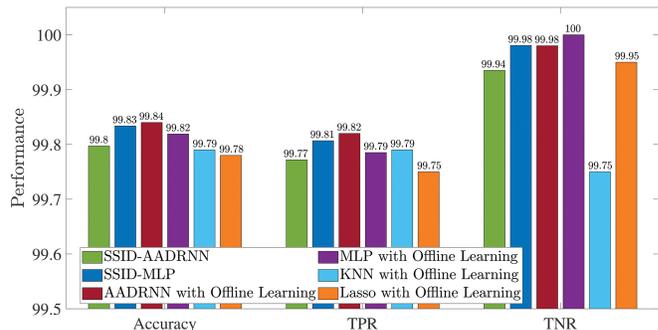


Fig. 8. Performance comparison between the ML models under the SSID framework and those with offline learning

D. Performance Evaluation for Compromised Device Identification

We now evaluate the performance of CDIS [8] (using the methodology in Section III) under the SSID framework, in short SSID-CDIS, on six different attacks, from the two distinct datasets Kitsune and Bot-IoT. For each dataset, the performance of SSID-CDIS is compared with the CDIS technique with sequential learning. Using the same methods as in [8], compromised device identification is performed for a 10 seconds long time window. The performance is evaluated using the Balanced Accuracy [66].

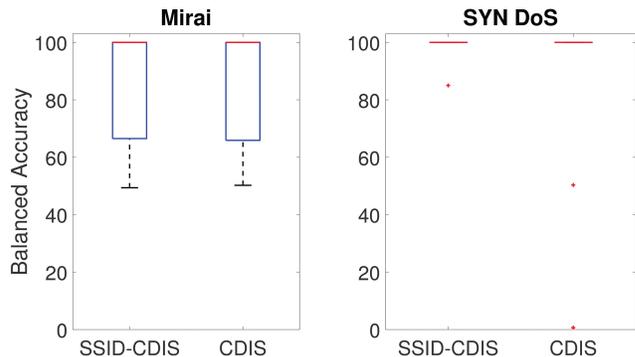


Fig. 9. Performance comparison of the CDIS trained under the SSID framework with that under sequential learning on Kitsune dataset

Figure 9 displays the performance of SSID-CDIS and its comparison with CDIS to identify compromised IP addresses, for each of the Mirai Botnet and SYN DoS attacks in the Kitsune dataset. Specifically it shows that while the SSID framework provides the same performance as sequential learning to identify compromised devices during a Mirai Botnet attack, it significantly improves the overall performance of CDIS during a SYN DoS attack. The box plot on the right of Figure 9 shows that SSID-CDIS achieves 100% median balanced accuracy when there is only one outlier IP address with around 85% accuracy. On the other hand, the sequentially

trained CDIS has two outlier IP addresses with performances of 50% and 1%, respectively.

TABLE II
COMPARISON OF AVERAGE PERFORMANCE OVER IP ADDRESSES BETWEEN SSID-CDIS AND DIFFERENT ML MODELS FOR MIRAI BOTNET FROM THE KITSUNE DATASET

Models	Balanced Accuracy	TPR	TNR
SSID-CDIS	89.1	60.6	87.7
CDIS	87.7	90.3	79.4
MLP	82.7	67.5	78
Lasso	85.1	86.7	75.6
KNN	82.1	74.4	74.1

In Table II, we present the average performance of the SSID-CDIS against each of Lasso, KNN, and MLP with respect to Balanced Accuracy, TPR and TNR on Kitsune Mirai dataset. The results in this table show that SSID-CDIS achieves much higher Balanced Accuracy and TNR than other models. Although its average TPR is considerably low, SSID-CDIS provides a reasonable compromised device identification accuracy with a much lower false alarm rate compared to other models.

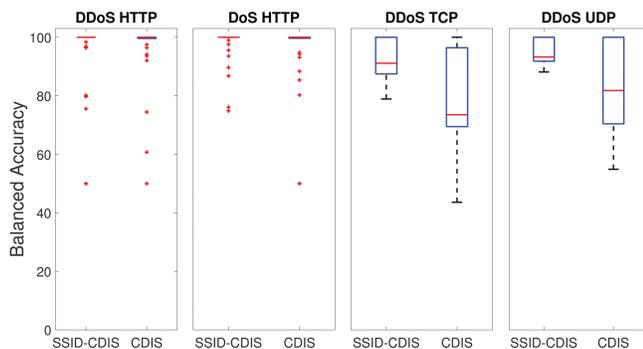


Fig. 10. Performance comparison of the CDIS trained under the SSID framework with that under sequential learning on Bot-IoT dataset

Figure 10 exhibits the performance of the SSID-CDIS system, and compares it with CDIS, to identify compromised IP addresses during DDoS and DoS attacks, using different communication protocols available in the Bot-IoT dataset. These results show that the SSID framework achieves higher identification performance, as compared to the use of CDIS sequential learning for the majority of attack types.

Starting with the box plot displayed at the far left of this figure, we observe the following results:

- 1) For the DDoS HTTP attack, the overall performance is almost the same for SSID and CDIS with sequential learning. However, as expected, performance varies slightly for individual IP addresses.
- 2) For the DoS HTTP attack, using SSID improved the performance by 2% on average with a minimum of 75% balanced accuracy.
- 3) For the DDoS TCP attack, SSID significantly improved the median accuracy by 18%, where SSID-CDIS

achieves 91% median accuracy. In addition, while the balance accuracy of CDIS with sequential learning is below 80% (with a minimum of 49%) for 9 out of 13 unique IP addresses, the balance accuracy of SSID-CDIS is equal to 79% for only 2 IP addresses and above 80% for the rest.

- 4) Similar to the results for the DDoS TCP attack, SSID was seen to provide significant performance improvement to identify compromised devices during a DDoS UDP attack. The median accuracy increased by 11%, achieving above 88% balanced accuracy for all IP addresses.

VI. CONCLUSION

This paper has proposed a novel Self-Supervised Intrusion Detection (SSID) framework which is designed to train any given IDS (whose parameters are calculated using the network traffic) **fully online** with no need for human intervention or prior offline training. The SSID framework comprises two successive learning stages, namely initial learning and online learning. Initial learning aims to quickly adapt the IDS parameters to the network where the IDS is deployed. Online learning aims to update the parameters whenever it is required to ensure high detection accuracy of the IDS.

During the real-time operation of the IDS, in parallel to detection, the SSID framework performs the following main tasks:

- It continually estimates the trustworthiness of intrusion decisions to identify normal and malicious traffic. It also measures the ability of the IDS to learn and generalize from data provided by SSID and the extent to which this data can represent the current online network traffic patterns.
- In order to provide training data for the IDS, the SSID framework selects and labels network traffic packets in a self-supervised manner based only on the decisions of IDS, and on the trust of SSID with regard to those decisions.
- The SSID framework determines when the IDS parameters need to be updated, based on the trustworthiness of the IDS, the selected training packets, and the latest state of network security.

Thus the proposed SSID framework eliminates the need for offline data collection, it prevents human errors in data labeling avoiding labor and computational costs for model training and data collection through experiments. Its most important advantage is in terms of performance, and it enables IDS to easily adapt to the time varying characteristics of the network traffic.

We also evaluated the performance of the SSID framework for two tasks: malicious traffic detection and compromised device identification to enhance the security of an IoT network. For malicious traffic detection, two different DL models, AADRNN and MLP, have been deployed with the SSID framework and tested on the Kitsune dataset. The results we obtain reveal that the DL models trained under the SSID framework without offline training also achieve considerably

high performance compared to the same models with offline and incremental learning.

For compromised device identification, the performance of the state-of-the-art CDIS has been tested under sequential learning and the SSID framework on data from six different cyberattacks provided by the two publicly available Kitsune and Bot-IoT datasets. The results show that the use of SSID significantly improves the performance of CDIS for the majority of cases considered.

Future work will evaluate the use of SSID for adapting a pre-trained IDS for use across different networks whose traffic has not been learned *a priori*, which seems to be a promising approach for fast, self-supervised, and successful adaptation of the IDS parameters for various networks. It would also be interesting to examine security assurance methods targeting distributed systems that combine the SSID framework with Federated Learning and attack prevention or mitigation algorithms. It seems that a successful integration of the SSID framework with Federated Learning may provide secure, distributed and self-supervised online learning for collaborative systems.

APPENDIX

Table III and Table IV respectively display the list of abbreviations and the list of symbols seen throughout this paper.

TABLE III
LIST OF ABBREVIATIONS (IN ALPHABETIC ORDER)

Abbreviation	Definition
AADRNN	Auto-Associative Deep Random Neural Network
AAM	Auto-Associative Memory
AE	Auto Encoder
ANOVA	Analysis of Variance
CDIS	Compromised Device Identification System
CNN	Convolutional Neural Network
DARPA	Defense Advanced Research Projects Agency
DDoS	Distributed Denial of Service
DL	Deep Learning
DNS	Domain Name System
DoS	Denial of Service
DRNN	Deep Random Neural Network
DT	Decision Tree
ELM	Extreme Learning Machine
FISTA	Fast Iterative Shrinkage-Thresholding Algorithm
IDS	Intrusion Detection System
IoT	Internet of Things
IP	Internet Protocol
ISSL	Incremental Semi-Supervised Learning
KL	Kullback-Leibler
KNN	K-Nearest Neighbour
Lasso	Least Absolute Shrinkage and Selector Operator
LR	Linear Regression
LSTM	Long-Short Term Memory
M2M	Machine-to-Machine
MitM	Man-in-the-Middle
ML	Machine Learning
MLP	Multi-Layer Perceptron
NB	Naive Bayes
RF	Random Forest
RNN	Random Neural Network
ROC	Receiver Operating Characteristic
SSID	Self-Supervised Intrusion Detection

TABLE IV
LIST OF SYMBOLS

Symbol	Definition
M	Total number of network traffic metrics which is equivalent to the number of inputs to the IDS
x_i^m	The m -th metric extracted from network traffic for packet i
x_i	The vector of input metrics, $x_i = [x_i^1, \dots, x_i^m, \dots, x_i^M]$
y_i	Binary output of the IDS indicating whether the packet i is malicious
\hat{x}_i	The output of DL-based AAM indicating the expected value of x_i in the absence of intrusion
W_h	Matrix of connection weights (including biases) between layer $h-1$ and h
W	Total number of learnable parameters in the ML model utilized in the IDS
$\zeta(\cdot)$	Activation function of a cluster in AADRNN
Γ	Trust coefficient indicating the trust of SSID on the decisions of the IDS
Θ	Threshold on the trust coefficient (i.e. minimum desired value of Γ)
B^l	Batch of packets selected for learning
K	Minimum number of packets to be learned in each learning phase
I	Length of window in terms of the number of packets to calculate average decision
γ	Intrusion threshold
p_i^-	Probability of selecting packet i to use as a benign packet
p_i^+	Probability of selecting packet i to use as a malicious packet
q_i	Probability of rejecting packet i to use in training
C_{rep}	Factor of representativeness of the traffic packets learned by IDS
C_{gen}	Factor of generalization ability of IDS
S_l^{TT}	Set of inter-transmission times of all packets learned by IDS until the end of learning phase l , which has mean of $1/\lambda_l$
S_l^{PL}	Set of packet lengths times of all packets learned by IDS until the end of learning phase l , which has mean of $1/\lambda_o$
S_o^{TT}	Set of inter-transmission times of all normal packets observed during continuous detection, which has mean of $1/\mu_l$
S_o^{PL}	Set of packet lengths times of all normal packets observed during continuous detection, which has mean of $1/\mu_o$
D_{KL}^{TT}	KL-Divergence from S_l^{TT} to S_o^{TT}
D_{KL}^{PL}	KL-Divergence from S_l^{PL} to S_o^{PL}
$D_{KL-norm}^{TT}$	Normalized KL-Divergence from S_l^{TT} to S_o^{TT}
$D_{KL-norm}^{PL}$	Normalized KL-Divergence from S_l^{PL} to S_o^{PL}
Δ	Adequacy of the packets learned by IDS
κ	Knowledge of IDS obtained from the packets learned
$\mathcal{E}(l)$	Empirical error measured at the end of learning phase l

REFERENCES

- [1] C. Douligeris and A. Mitrokotsa, "DDoS attacks and defense mechanisms: classification and state-of-the-art," *Computer networks*, no. 5, pp. 643–666, 2004.
- [2] D. Goodin, "100,000-strong Botnet built on router 0-day could strike at any time," *Ars Technica*, December 2017. [Online]. Available: <https://arstechnica.com/information-technology/2017/12/100000-strong-botnet-built-on-router-0-day-could-strike-at-any-time/>
- [3] Imperva, "2022 Imperva Bad Bot Report," p. 1–37, 2022. [Online]. Available: <https://www.imperva.com/resources/resource-library/reports/bad-bot-report/>
- [4] B. Tushir, H. Sehgal, R. Nair, B. Dezfouli, and Y. Liu, "The impact of dos attacks on resource-constrained iot devices: A study on the mirai attack," *arXiv preprint arXiv:2104.09041*, 2021.

- [5] Cisco, *Cisco Annual Internet Report (2018–2023)*, Mar. 2020. [Online]. Available: <https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html>
- [6] H. Liu and B. Lang, “Machine learning and deep learning methods for intrusion detection systems: A survey,” *applied sciences*, vol. 9, no. 20, p. 4396, 2019.
- [7] M. Nakip and E. Gelenbe, “Botnet attack detection with incremental online learning,” in *Security in Computer and Information Sciences: Second International Symposium, EuroCybersec 2021, Nice, France, October 25–26, 2021, Revised Selected Papers*. Springer, 2022, pp. 51–60.
- [8] E. Gelenbe and M. Nakip, “Traffic based sequential learning during botnet attacks to identify compromised IoT devices,” *IEEE Access*, vol. 10, pp. 126 536–126 549, 2022.
- [9] H. M. Song and H. K. Kim, “Self-supervised anomaly detection for in-vehicle network using noised pseudo normal data,” *IEEE Transactions on Vehicular Technology*, vol. 70, no. 2, pp. 1098–1108, 2021.
- [10] Z. Wang, Z. Li, J. Wang, and D. Li, “Network intrusion detection model based on improved byol self-supervised learning,” *Security and Communication Networks*, vol. 2021, pp. 1–23, 2021.
- [11] X. Zhang, J. Mu, X. Zhang, H. Liu, L. Zong, and Y. Li, “Deep anomaly detection with self-supervised learning and adversarial training,” *Pattern Recognition*, vol. 121, p. 108234, 2022.
- [12] H. Kye, M. Kim, and M. Kwon, “Hierarchical detection of network anomalies: A self-supervised learning approach,” *IEEE Signal Processing Letters*, vol. 29, pp. 1908–1912, 2022.
- [13] E. Caville, W. W. Lo, S. Layeghy, and M. Portmann, “Anomal-e: A self-supervised network intrusion detection system based on graph neural networks,” *Knowledge-Based Systems*, vol. 258, p. 110030, 2022.
- [14] W. Wang, S. Jian, Y. Tan, Q. Wu, and C. Huang, “Robust unsupervised network intrusion detection with self-supervised masked context reconstruction,” *Computers & Security*, vol. 128, p. 103131, 2023.
- [15] M. Abououf, R. Mizouni, S. Singh, H. Otrok, and E. Damiani, “Self-supervised online and lightweight anomaly and event detection for IoT devices,” *IEEE Internet of Things Journal*, vol. 9, no. 24, pp. 25 285–25 299, 2022.
- [16] B. H. Meyer, A. T. Pozo, M. Nogueira, and W. M. Numan Zola, “Federated self-supervised learning for intrusion detection,” in *2023 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2023, pp. 822–828.
- [17] “Kitsune Network Attack Dataset,” August 2020. [Online]. Available: <https://www.kaggle.com/ymirsky/network-attack-dataset-kitsune>
- [18] N. Koroniotis, N. Moustafa, E. Sitnikova, and B. Turnbull, “Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-IoT dataset,” *Future Generation Computer Systems*, vol. 100, pp. 779–796, 2019. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0167739X18327687>
- [19] T. A. Tuan, H. V. Long, R. Kumar, I. Priyadarshini, N. T. K. Son *et al.*, “Performance evaluation of botnet ddos attack detection using machine learning,” *Evolutionary Intelligence*, pp. 1–12, 2019.
- [20] Z. Shao, S. Yuan, and Y. Wang, “Adaptive online learning for IoT botnet detection,” *Information Sciences*, vol. 574, pp. 84–95, 2021.
- [21] M. Shafiq, Z. Tian, A. K. Bashir, X. Du, and M. Guizani, “Corrauc: A malicious Bot-IoT traffic detection method in IoT network using machine-learning techniques,” *IEEE Internet of Things Journal*, vol. 8, no. 5, pp. 3242–3254, 2021.
- [22] R. Doshi, N. Aphorpe, and N. Feamster, “Machine learning ddos detection for consumer internet of things devices,” in *2018 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2018, pp. 29–35.
- [23] I. Letteri, M. Del Rosso, P. Caianiello, and D. Cassioli, “Performance of botnet detection by neural networks in software-defined networks,” in *ITASEC*, 2018.
- [24] M. Banerjee and S. Samantaray, “Network traffic analysis based IoT botnet detection using honeynet data applying classification techniques,” *International Journal of Computer Science and Information Security (IJCSIS)*, vol. 17, no. 8, 2019.
- [25] C. D. McDermott, F. Majdani, and A. V. Petrovski, “Botnet detection in the Internet of Things using deep learning approaches,” in *2018 international joint conference on neural networks (IJCNN)*. IEEE, 2018, pp. 1–8.
- [26] C. Tzagarakis, N. Petroulakis, and S. Ioannidis, “Botnet attack detection at the IoT edge based on sparse representation,” in *2019 Global IoT Summit (GloTS)*. IEEE, 2019, pp. 1–6.
- [27] Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, A. Shabtai, D. Breitenbacher, and Y. Elovici, “N-baiot—network-based detection of IoT botnet attacks using deep autoencoders,” *IEEE Pervasive Computing*, vol. 17, no. 3, pp. 12–22, 2018.
- [28] C. S. Htwe, Y. M. Thant, and M. M. S. Thwin, “Botnets attack detection using machine learning approach for IoT environment,” in *Journal of Physics: Conference Series*, vol. 1646, no. 1. IOP Publishing, 2020, p. 012101.
- [29] S. Sriram, R. Vinayakumar, M. Alazab, and K. Soman, “Network flow based IoT botnet attack detection using deep learning,” in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2020, pp. 189–194.
- [30] Y. N. Soe, Y. Feng, P. I. Santosa, R. Hartanto, and K. Sakurai, “Machine learning-based IoT-botnet attack detection with sequential architecture,” *Sensors*, vol. 20, no. 16, p. 4372, 2020.
- [31] G. D. L. T. Parra, P. Rad, K.-K. R. Choo, and N. Beebe, “Detecting Internet of Things attacks using distributed deep learning,” *Journal of Network and Computer Applications*, vol. 163, p. 102662, 2020.
- [32] J. Liu, S. Liu, and S. Zhang, “Detection of IoT botnet based on deep learning,” in *2019 Chinese Control Conference (CCC)*. IEEE, 2019, pp. 8381–8385.
- [33] G. Bovenzi, G. Aceto, D. Ciunzo, A. Montieri, V. Persico, and A. Pescapé, “Network anomaly detection methods in IoT environments via deep learning: A fair comparison of performance and robustness,” *Computers & Security*, vol. 128, p. 103167, 2023.
- [34] S. Garcia, A. Parmisano, and M. J. Erquiaga, “IoT-23: A labeled dataset with malicious and benign IoT network traffic,” May 2021. [Online]. Available: <https://doi.org/10.5281/zenodo.4743746>
- [35] A. Kumar and T. J. Lim, “Early detection of mirai-like iot bots in large-scale networks through sub-sampled packet traffic analysis,” in *Future of Information and Communication Conference*. Springer, 2019, pp. 847–867.
- [36] M. Chatterjee, A. S. Namin, and P. Datta, “Evidence fusion for malicious bot detection in iot,” in *2018 IEEE International Conference on Big Data (Big Data)*, 2018, pp. 4545–4548.
- [37] T. D. Nguyen, S. Marchal, M. Miettinen, H. Fereidooni, N. Asokan, and A.-R. Sadeghi, “Dĭot: A federated self-learning anomaly detection system for iot,” in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, 2019, pp. 756–767.
- [38] N. V. Abhishek, T. J. Lim, B. Sikdar, and A. Tandon, “An intrusion detection system for detecting compromised gateways in clustered iot networks,” in *2018 IEEE International Workshop Technical Committee on Communications Quality and Reliability (CQR)*. IEEE, 2018, pp. 1–6.
- [39] M. Taneja, “An analytics framework to detect compromised iot devices using mobility behavior,” in *2013 International Conference on ICT Convergence (ICTC)*, 2013, pp. 38–43.
- [40] A. O. Prokofiev, Y. S. Smirnova, and V. A. Surov, “A method to detect internet of things botnets,” in *2018 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EICon-Rus)*. IEEE, 2018, pp. 105–108.
- [41] T. N. Nguyen, Q.-D. Ngo, H.-T. Nguyen, and G. L. Nguyen, “An advanced computing approach for iot-botnet detection in industrial internet of things,” *IEEE Transactions on Industrial Informatics*, vol. 18, no. 11, pp. 8298–8306, 2022.
- [42] A. Hristov and R. Trifonov, “A model for identification of compromised devices as a result of cyberattack on iot devices,” in *2021 International Conference on Information Technologies (InfoTech)*, 2021, pp. 1–4.
- [43] T. Trajanovski and N. Zhang, “An automated and comprehensive framework for iot botnet detection and analysis (iot-bda),” *IEEE Access*, vol. 9, pp. 124 360–124 383, 2021.
- [44] H. Bahşı, S. Nömm, and F. B. La Torre, “Dimensionality reduction for machine learning based iot botnet detection,” in *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, 2018, pp. 1857–1862.
- [45] J.-B. Grill, F. Strub, F. Alché, C. Tallec, P. Richemond, E. Buchatskaya, C. Doersch, B. Avila Pires, Z. Guo, M. Gheshlaghi Azar *et al.*, “Bootstrap your own latent—a new approach to self-supervised learning,” *Advances in neural information processing systems*, vol. 33, pp. 21 271–21 284, 2020.
- [46] X. Liu, F. Zhang, Z. Hou, L. Mian, Z. Wang, J. Zhang, and J. Tang, “Self-supervised learning: Generative or contrastive,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 1, pp. 857–876, 2023.
- [47] J. Yu, H. Yin, X. Xia, T. Chen, J. Li, and Z. Huang, “Self-supervised learning for recommender systems: A survey,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 36, no. 1, pp. 335–355, 2024.
- [48] E. Gelenbe and Y. Yin, “Deep learning with random neural networks: I,” in *2016 International Joint Conference on Neural Networks (IJCNN)*, 2016, pp. 1633–1638.

- [49] E. Gelenbe and Y. Yin, "Deep learning with dense random neural networks: II," in *International Conference on Man-Machine Interactions*. Springer, 2017, pp. 3–18.
- [50] E. Gelenbe, "Random neural networks with negative and positive signals and product form solution," *Neural Computation*, vol. 1, no. 4, pp. 502–510, 1989.
- [51] —, "Learning in the recurrent random neural network," *Neural Computation*, vol. 5, no. 1, pp. 154–164, 1993.
- [52] O. Brun et al., "Deep learning with dense random neural networks for detecting attacks against IoT-connected home environments: I," in *International ISCIS Cyber-Security Workshop*. Springer, Cham, 2018, pp. 79–89.
- [53] O. Brun, Y. Yin, and E. Gelenbe, "Deep learning with dense random neural network for detecting attacks against iot-connected home environments: II," *Procedia Computer Science*, vol. 134, pp. 458–463, 2018.
- [54] M. Nakip and E. Gelenbe, "Mirai botnet attack detection with auto-associative dense random neural network," in *2021 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2021, pp. 01–06.
- [55] E. Gelenbe and M. Nakip, "G-networks can detect different types of cyberattacks," in *2022 30th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 2022, pp. 9–16.
- [56] E. Gelenbe, "G-networks: a unifying model for neural and queueing networks," *Annals of Operations Research*, vol. 48, no. 5, pp. 433–461, 1994.
- [57] S. Evmorfos et al., "Neural network architectures for the detection of syn flood attacks in IoT systems," in *Proceedings of the 13th ACM International Conference on Pervasive Technologies Related to Assistive Environments*, 2020, pp. 1–4.
- [58] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM journal on imaging sciences*, vol. 2, no. 1, pp. 183–202, 2009.
- [59] N.-y. Liang, G.-b. Huang, P. Saratchandran, and N. Sundararajan, "A fast and accurate online sequential learning algorithm for feedforward networks," *IEEE Transactions on Neural Networks*, vol. 17, no. 6, pp. 1411–1423, 2006.
- [60] S. Kullback and R. A. Leibler, "On information and sufficiency," *The annals of mathematical statistics*, vol. 22, no. 1, pp. 79–86, 1951.
- [61] O. Bousquet, S. Boucheron, and G. Lugosi, "Introduction to statistical learning theory," *Advanced Lectures on Machine Learning: ML Summer Schools 2003, Canberra, Australia, February 2-14, 2003, Tübingen, Germany, August 4-16, 2003, Revised Lectures*, pp. 169–207, 2004.
- [62] A. Alwosheel, S. van Cranenburgh, and C. G. Chorus, "Is your dataset big enough? sample size requirements when using artificial neural networks for discrete choice analysis," *Journal of choice modelling*, vol. 28, pp. 167–182, 2018.
- [63] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An ensemble of autoencoders for online network intrusion detection," in *The Network and Distributed System Security Symposium (NDSS) 2018*, 2018.
- [64] N. Moustafa and J. Slay, "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," in *2015 Military Communications and Information Systems Conference (MilCIS)*, 2015, pp. 1–6.
- [65] I. Sharafaldin, A. H. Lashkari, A. A. Ghorbani et al., "Toward generating a new intrusion detection dataset and intrusion traffic characterization," *ICISSp*, vol. 1, pp. 108–116, 2018.
- [66] K. H. Brodersen, C. S. Ong, K. E. Stephan, and J. M. Buhmann, "The balanced accuracy and its posterior distribution," in *2010 20th international conference on pattern recognition*. IEEE, 2010, pp. 3121–3124.



Mert NAKIP obtained his B.Sc. degree, graduating with the first rank in his class, and subsequently completed his M.Sc. thesis in Electrical-Electronics Engineering at Yaşar University (Izmir, Turkey) in 2018 and 2020, respectively. His design of a multi-sensor fire detector utilizing ML achieved the national #1 ranking at the Industry-Focused Undergraduate Graduation Projects Competition organized by TÜBİTAK (Turkish Scientific and Technological Research Council). His M.Sc. thesis, focusing on applying machine learning techniques to Massive Access of IoT, supported by the National Graduate Scholarship Program of TÜBİTAK 2210C in High-Priority Technological Areas. He received his Ph.D. from the Institute of Theoretical and Applied Informatics, Polish Academy of Sciences (Gliwice, Poland) in January 2024, and is now serving as an Assistant Professor there. He has participated in projects funded by TÜBİTAK and the European Commission concerning IoT, cybersecurity, and machine learning. Currently, he is involved in the DOSS project.



Erol GELENBE FIEEE, FACM, FIFIP, FRSS, FIET, graduated from METU (Turkey), and received the PhD from Polytechnic Institute of NYU, and the DSc from Sorbonne University. He pioneered system performance evaluation methods, and invented the Random Neural Network, while helping to develop commercial products including the Queueing Network Analysis Package and the manufacturing simulator FLEXSIM. He has graduated over 90 PhDs including 94 women. Professor at the Institute of Theoretical and Applied Informatics, Polish Academy of Sciences, he previously held chairs at Imperial College, University of Central Florida, Duke University, Université Paris-Descartes and Paris-Saclay, and Université de Liège. His prizes include the Parlar Foundation Science Award (Turkiye), Grand Prix France Télécom, the ACM SIGMETRICS Life-Time Achievement Award, the UK IET Oliver Lodge Medal and the Mustafa Prize. Elected Fellow of Academia Europaea, the French National Acad. of Technologies, the Science Academies of Hungary, Poland, Turkey and the Royal Acad. of Belgium. He was awarded Chevalier de la Légion d'Honneur, Chevalier des Palmes Académiques and Commandeur du Mérite by France, Commander of the Order of the Crown of Belgium, Commendatore al Merito and Grande Ufficiale dell'Ordine della Stella by Italy, and Officer of the Order of Merit of Poland. Principal Investigator of numerous European Union research projects, Coordinator of FP7 NEMESYS and H2020 SerIoT, he has also been supported by the US NSF, ONR, ARO, the UKRI and industry.