World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:17, No:2, 2023

Real-time Network Anomaly Detection Systems Based on Machine-Learning Algorithms

Zahra Ramezanpanah, Joachim Carvallo, Aurelien Rodriguez

Abstract—This paper aims to detect anomalies in streaming data using machine learning algorithms. In this regard, we designed two separate pipelines and evaluated the effectiveness of each separately. The first pipeline, based on supervised machine learning methods, consists of two phases. In the first phase, we trained several supervised models using the *UNSW-NB15* data set. We measured the efficiency of each using different performance metrics and selected the best model for the second phase. At the beginning of the second phase, we first, using Argus Server, sniffed a local area network. Several types of attacks were simulated and then sent the sniffed data to a running algorithm at short intervals. This algorithm can display the results of each packet of received data in real-time using the trained model. The second pipeline presented in this paper is based on unsupervised algorithms, in which a Temporal Graph Network (TGN) is used to monitor a local network. The TGN is trained to predict the probability of future states of the network based on its past behavior. Our contribution in this section is introducing an indicator to identify anomalies from these predicted probabilities.

Keywords—Cyber-security, Intrusion Detection Systems, Temporal Graph Network, Anomaly Detection.

I. Introduction

TODAY, life without the Internet is inconceivable, and this use of the Internet is not limited to spending time on social media and entertainment. Today, we need the Internet to do our daily work. According to [24], since the beginning of 2021, there have been 4.66 billion in active Internet users, and this number represents an increase of 316 million compared to the 2020 figures. In other words, the growth of active Internet users worldwide (7.3% percent) is more than seven times faster than the growth of the total population, which is 1%. With the pervasiveness of the Internet in everyday life, especially in storing personal, confidential, and essential information of individuals and companies on servers, the urgent need to automatically detect attacks and unusual events (intrusion) in networks has become essential. In this regard, Intrusion Detection Systems (IDS) technology, which is hardware or software programs that monitor network activities to detect malicious behavior or policy violations, has emerged and attracted much attention. IDS have four techniques for performing their tasks, [1]: Signature-based, Anomaly-based, Specification-based, and hybrid. Signature-based techniques can identify anomalies whose specifications have already been documented. Anomaly-based techniques consist in collecting data and detecting system anomalies based on a threshold value, and they are still considered to be at an early stage of development. Those methods effectively detect unknown and unseen attacks, but the need for large memory, inducing high processing costs, constitutes an important limitation to their prevalence. Specification-based techniques continuously evaluate system performance. In this approach, network administrators define specific policies, and the processes are constantly monitored to make sure they respect the policies. Hybrid-based techniques combine anomalies and signature-based methods to better trade-off between storage and calculation costs with fewer false-positive alerts. In this paper, we used a Signature-based technique to design the first pipeline and an Anomaly-based technique to design the second pipeline. Using these two pipelines, the contributions of this study are: (1) Deploy an offline to an online pipeline to detect anomalies in real-time; (2) Introducing a new indicator to detect anomalies using a Temporal Graph Network.

Zahra Ramezanpanah is with MYTEAM.ai, France (e-mail: zahra.ramezanpanah@gmail.com).

II. Background and Related Works

Numerous studies have been conducted in cyber security to detect various types of anomalies and cyberattacks or intrusions. Among the available methods for detecting anomalies and preventing cyberattacks, Signature-based intrusion detection systems (SIDS) and Anomaly-based intrusion detection systems (AIDS) are the most efficient [3]. SIDS are formed on the signatures of known anomalies [5]. AIDS, on the other hand, can detect previously unseen anomalies that are invisible to SIDS [6], [7]. Since machine learning classification-based methods can automatically learn from network data [8] we first focus on these methods. Different machine learning methods have been used for different purposes in this topic. For example, the authors in [9] detected anomalies in a network using Support Vector Machine (SVM) classification. They achieved 98.62% accuracy by implementing their proposed method on the KDD cup 99 data set [10]. Anomalies detection by SVM was also used by many other researchers such as [11], [12]. K-Nearest Neighbor (KNN) is another machine learning method that stores the specifications of all input data and classifies new data based on its similarity to existing data. The authors in [13], [14] used KNN to classify network attacks in their research. In [15], Restricted Boltzmann Machine (RBM) is used to reduce the features, and SVM is implemented to identify the anomalies. This method has an accuracy of 87%. Subsequently, while many studies are using ML and DL algorithms in the design of IDS, to our knowledge, the real-time application of these algorithms on a network has not been well discussed.

We review below some current research work on recognizing anomalies in real-time. In order to reduce network data processing time, the authors in [16] proposed a real-time

IDS using a multi-operating system (MAS-IDS). They succeed in analyzing a large volume of data in a network in the shortest possible time. For this purpose, they divided the traffic data network into several subsets and processed these subsets in parallel, using a set of analysis agents. Another important conclusion of their proposed system was reducing the time for recognizing anomalies. At the same time, the accuracy of IDS was acceptable and was not much different from the traditional method, which does not include division. In order to improve the performance of anomaly detection, in [17], deep learning (DL) is used in the combustion anomaly detection system. The researchers in this paper used deep learning to learn the hierarchical features of an exhaust gas temperature sensor. They then used these features as the input of a neural network classifier to detect combustion anomalies. Since these features highlight the complex relationships between all sensor measurements, abnormalities detection is significantly more accurate. De et al. The authors in [18], used a Graph Neural Network (GNN) mechanism and fully connected networks to create an edge and node classifier. This mechanism leads to the probability of infection in a relevant node and vertices. Analyzing graph representations, they exchanged information in the neighborhood of an agent that helped identify anomalies and security breaches based on their interactions. They used the response of a node to the health status of its neighboring nodes to solve the problem of an inadequate number of monitors against distributed attack patterns and the glands' lack of awareness of their infected status. At the same time, they sought to reduce resources such as bandwidth and power consumption compared to centralized IDS. Each active node applies GNNs to connection attribute values in its local neighborhood. To train and test GNN, they used real-world data sets of normal traffic patterns and, in addition, generated data based on the Mirai's [19] anomalous distribution. They used three different classifiers (SVM, DT, RT) and two advanced approaches to train and test generated data, thus evaluating the accuracy scores of their proposed method.

In this paper, we monitor the network and identify anomalies in real-time using supervised (first system) and unsupervised (second system) ML methods. For this purpose, in the first system, using the information available in studies conducted in the literature, we selected the best parameters for training the model in the online phase. In the second system, by presenting an index and using the proposed method in [20], we were able to monitor a simulated network and detect anomalies.

III. Machine Learning Algorithms and Performance Metrics

A. Methods

In this paper, we trained five algorithms [25], namely Support Vector Machine (SVM), Decision Tree (DT), Random Forest (RF), Gradient Boosting Decision Tree (GBDT), and Logistic Regression (LR) to design the first system based on supervised ML algorithms. To design the second system, we used a Temporal Graph Network (TGN), whose details have changed a bit, and we will explain it below.

1) Temporal Graph Networks (TGN): [20] are an efficient framework for deep learning on "continuous-time dynamic graphs". This title refers to graphs that can be represented as a sequence of time-stamped events, such as adding or removing an edge or a node. In this paper, nodes represent machines in a network whose number is constant, and edges represent network traffic (packets exchanged between machines) and are therefore constantly changing over time. A neural model for dynamic graphs can be defined as an encoder-decoder pair in which the encoder is a function that maps the node embedding (a vector of important information of nodes) [21], from a dynamic graph. A decoder receives one or more node embedding as input and performs its task based on these inputs. In our case, the decoder task is to calculate the probability of interaction between two nodes at any given time. A TGN consists of the following modules:

False Message Generator (FMG): For each batch containing $p \in \mathbb{N}$ interactions, this module randomly generates p false interactions. For each interaction on the network, t, s, and d represent the time, the source node, and the destination node. This module is not explicitly described in [20] because the paper does not focus on the task of future edge prediction. We decided here to give it a name for better clarity. In [20]], this module works as follows: duplicate the interactions of the current batch and randomly modify the destinations d. We consider in this work that this module is more general and that any false message generation can be considered.

Message Function (MSG): This module generates two messages from each interaction in the batch: one for the source and one for the destination.

Message Aggregator (AGG): This module handles the cases where the same node has multiple messages. There are several ways to solve this issue, such as considering only the most recent message or considering the mean of duplicated messages.

Memory (S): This module contains a vector $s_i(t)$ for each machine in the network. This memory vector, which represents the history of each node, is updated after each batch using the Memory updater.

Memory Updater (MEM): This module takes as input the messages and the Memory to update the Memory.

Embedding (EMB): This module computes embeddings for each node at any time t and its main purpose is to prevent the problem of memory staleness. [22].

Decoder (DEC): This module (in our case) takes as input the embeddings of the source and destination of interaction and classify the interaction between false and real.

Raw Messages Store (RMS): This module is a container for the batch at the end of a training step.

In order to train a TGN, we detail the procedure in Algorithm 1.

B. Performance Metrics

We evaluated the performance of the supervised ML algorithms by 3 criteria: Accuracy, False Alarm Rate, and

Algorithm 1 Train a TGN

Input: $\{B^{(i)} = (t^{(i)}, s^{(i)}, d^{(i)}) = \{(t_j^{(i)}, s_j^{(i)}, d_j^{(i)})\}_{0 \leq j \leq p}\}$: Training data

$[S]S \leftarrow 0$ ▷ Initialize memory to zeros
$[RMS]RMS \leftarrow \{\}$ ▷ Initialize raw messages
for each $B^{(i)} \in$ training data **do**
  $B_{neg}^{(i)} \leftarrow [FMG]FMG(B^{(i)})$ ▷ Sample fasle interactions
  $m \leftarrow [MSG]MSG([RMS]RMS)$ ▷ Compute messages from the previous batch
  $\bar{m} \leftarrow [AGG]AGG(m)$ ▷ Aggregate messages for the same node
  $\hat{S} \leftarrow [MEM]MEM(\bar{m}, S)$ ▷ Get updated memory
  $z \leftarrow [EMB]EMB_{\hat{S}}([B^{(i)}, B_{neg}^{(i)}])$ ▷ Compute embeddings
  $p^{(i)}, p_{neg}^{(i)} \leftarrow [DEC]DEC(z)$ ▷ Compute interaction probabilities
  $loss \leftarrow$ *binary cross-entropy*$(p^{(i)}, p_{neg}^{(i)})$
  $[RMS]RMS \leftarrow B^{(i)}$ ▷ Store the batch
  $[S]S \leftarrow \hat{S}$ ▷ Update the memory
end for

F1-Score derived from the confusion matrix [23].

IV. Proposed Systems

A. Supervised Prediction

As shown in Fig. 1, we need a trained model to identify anomalies in real-time. For this purpose, using 70% of a public data set called UNSW-NB15[4], we trained 5 ML algorithms and selected the one with the best results for use in the online phase. In order to use the training data as input to the algorithms, the data must first go through the preparation steps, which include removing highly correlated features, adding new features (Total bytes transferred by the network), and standardizing by applying *standardscaler*. After this, all the features will have mean 0 and std 1.

The algorithms trained by this processed data were then evaluated by the remaining 30% of the data set using the confusion matrix and other criteria described in Section III-B. Then in the next step, to identify the anomalies in real-time, we need to run the four functions parallel as follows.

Sniffing function: sniffs the network traffic using Argus server[1] in real time.

Feature extraction function: extracts the same features that we used in the offline phase from the sniffed network using Argus.

Pre-processing function: prepares features as the inputs for next function.

Predictions function: classifies the network flow in real time. All labels predicted by the model are recorded in the data set.

B. Unsupervised Prediction

This section presents our method for performing unsupervised anomaly detection on a network using a TGN and our new indicator. The steps are as follows:

1) Training of a TGN using algorithm 1, on a network behaving normally (without anomalies). After training the model, one can interpret its prediction for an interaction at a given time t as an estimate of the probability that this interaction occurs in the network (assuming normal network behavior).
2) Estimation of the probability of every new interaction (t_i, s_i, d_i) occurring on the network, using the trained TGN (with its memory up-to-date with all messages that occurred before t_i). We obtain an estimate of the sequence of probabilities of consecutive interactions on our network $p_0, p_1, ..., p_n$ at times $t_0, t_1, ..., t_n \leq T$.
3) Computation of the indicator $A(T)$ (1), which represents the "anomaly level" at time T. The calculation of this indicator is done iteratively by dividing the time into fixed intervals of size Δt. This indicator associates successive low probabilities with a more abnormal network state.

$$A(t+\Delta t) = A(t) \times \beta + \sum_{t \leq t_i \leq t+\Delta t} \frac{1}{(p_{e(t_i)} + \epsilon)^{\alpha}} - 1 \quad (1)$$

in which $A(0) = 0$, $\beta \in]0, 1[$ controls the speed of the exponential decay of the indicator, $\alpha \in R^+$ controls the amplitude of the jumps induced by the right-hand side of (1) and $0 \leq \epsilon$ prevents the division by zero problem and reduces the importance of probabilities very close to zero.
4) Setting a threshold so that if $A(T)$ exceeds it, an alarm is triggered. The choice of this threshold monitors the compromise between the number of false alarms and the rate of correctly detected anomalies.

The outline of this process is summarized in the following algorithm 2.

After employing Algorithm 2 to evaluate anomaly level on the network at time T, through the indicator $A(T)$, it is not necessary to redo all the calculations to obtain A again at a later time $T_{new} > T$. Instead, Algorithm 2 can be re-applied with the new interactions $\{(t_i, s_i, d_i) : T < t_i < T_{new}\}$ as input and the memory S, the indicator A and the time t_{new} from the last iteration should not be re-initialized to zero.

[1] http://qosient.com/argus/index.shtml

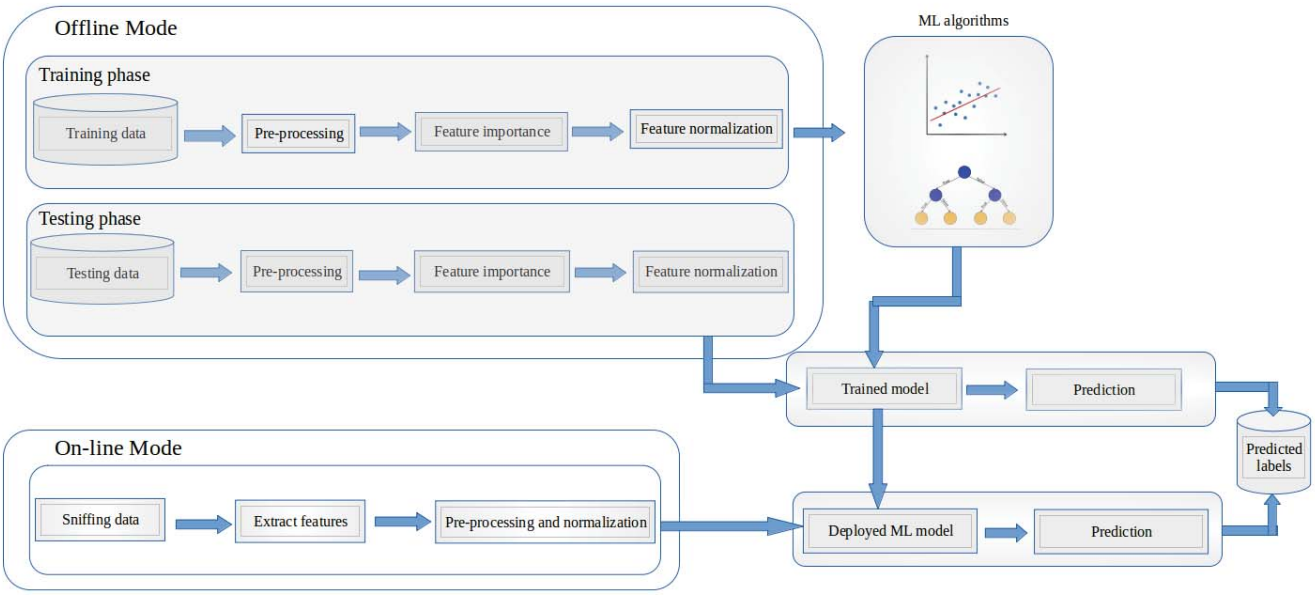

Fig. 1 Supervised machine learning-based intrusion detection system pipeline

Algorithm 2 Evaluation phase

Input: $\{(t_0, s_0, d_0), \ldots, (t_i, s_i, d_i), \ldots, (t_n, s_n, d_n)\}$, $t_i < T$: Current time
Output: 0 or 1

$[S]S \leftarrow 0$
$[RMS]RMS \leftarrow \{\}$
for each (t_i, s_i, d_i) **do**
  $m \leftarrow [MSG]MSG([RMS]RMS)$
  $\hat{S} \leftarrow [MEM]MEM(m, S)$
  $z_{s_i}, z_{d_i} \leftarrow [EMB]EMB_{\hat{S}}(s_i, t_i), [EMB]EMB_{\hat{S}}(d_i, t_i)$
  $p_i \leftarrow [DEC]DEC(z_{s_i}, z_{d_i})$
  $[RMS]RMS \leftarrow \{(t_i, s_i, d_i)\}$
  $[S]S \leftarrow \hat{S}$
end for
$[indicator]A \leftarrow 0$
$t_{new} \leftarrow 0$
while $t_{new} < T$ **do**
  $[indicator]A \leftarrow [indicator]A \times \beta + \sum_{t_{new} \leq t_i \leq t_{new} + \Delta t} \frac{1}{(p_i + \epsilon)^{\alpha}} - 1$
  $t_{new} \leftarrow t_{new} + \Delta t$
end while
if $[indicator]A > threshold$ **then**
  return 1
else
  return 0
end if

V. Experimental Results and Discussion

A. Supervised Prediction

The first step in this system is data preparation. There are 5 categorical columns with text data which can not be processed by ML algorithms: 1. *srcip*, 2. *dstip*, 3. *proto*, 4 *state* and 5. *service*. This data set is a public one. Our goal in this article is to identify anomalies in real-time, which our local office network evaluated, so the first two features (source IP and destination IP) that change according to the used network were removed. In order to convert the remaining three columns to numeric columns, we used a one-hot encoder, which assigns one if the value is available for the row, and the rest of the columns will be 0. Afterward, we removed the features with a high correlation with other features. Highly correlated features do not necessarily worsen the model, but they do not improve it either. So in order to reduce the computation time, which is very important in real-time detection, we removed them. To do this, we used the Pearson Correlation coefficient, r, to calculate the degree of similarity between two features according to the following formula:

$$r = \frac{\sum_{i=1}^{n}(x_i - \tilde{x})(y_i - \tilde{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \tilde{x})^2 (y_i - \tilde{y})^2}} \quad (2)$$

where $\tilde{x}$ and $\tilde{y}$ represents the mean of all records in features x and y respectively. The Pearson Correlation returns a value from -1 to 1. Proximity of this value to 1 indicates a high degree of correlation between the two features. In the next step, we added the total bytes transferred by the network. This feature represents the sum of 'sbytes' and 'dbytes'. In order to reduce the real-time calculations, we used only one *ZEEK* to extract the features, so we removed the properties that *ZEEK* could not extract from the data, and finally kept only the features *dttl*, *swin*, *stime*, *tcprtt*, *synack*, *ackdat*, *dur*, *sbytes*, *dbytes*, *sload*, *dload*, *spkts*, *stcpb*, *dtcpb*, *smeansz*, *dmeansz*, *network_bytes*, *proto* and *state*. Given that the *proto* and state

features using onehot method have been converted to Boolean data, 169 features remain for training at the end. Finally, we trained LR, Linear SVC, DT, RF and GBDT and calculated accuracy, F1-score and False Alarm Rate (FAR) for training and testing data, the results of which are shown in Table I.

TABLE I
SUPERVISED CCLASSIFICATION RESULTS

MODEL	DATA SET	ACCURACY	F1-SCORE	FAR
LR	Train-Test	0.99-0.99	0.95-0.95	0.0086-0.0087
SVC	Train-Test	0.99-0.99	0.95-0.95	0.009-0.009
DT	Train-Test	0.98-0.98	0.96-0.96	0.019-0.019
RF	Train-Test	0.99-0.98	0.987-0.980	0.007-0.010
GBDT	Train-Test	0.99-0.99	0.99-0.98	0.003-0.008

According to the results in Table I, it can be concluded that by using LR, we can have a model that has the highest accuracy, the lowest false alarm rate and without having the problem of overfitting. In the next step, we upgraded this model to an online model and used sniffed data from our local network as test data. We selected a machine as the attacking machine and sent the port scanning packets to another machine using *Nmap* [2] and sniffed the data at that time. The designed algorithm detected the anomaly in real-time, and the data processing time was less than 1 second. In order to obtain the accuracy of the algorithm in identifying the anomalies in the online mode, we labeled the data over 5 minutes. All packets sent from the attacking machine to the victim machine were labeled as an attack in this labeling. Table II shows the results obtained.

TABLE II
ONLINE CLASSIFICATION RESULTS

MODE	DATA SET	ACCURACY	F1-SCORE	FAR
Online	Test	0.95	0.91	0.049

B. Unsupervised Prediction

1) Data: In this system, our focus is on recognizing anomalies that could come from any machine within a network. Public data sets are often designed for only a few machines with fixed addresses responsible for carrying out attacks. For this reason, in this section, we simulated our network to provide the required data as follows:

Normal Network: This network consists of 10 machines whose interaction with each other is controlled by node-dependent rules. Fig. 2 shows the average of all interactions formed between every two machines on the network, where blue nodes represent the machines and black arrows represent the interaction between them.

The interactions generated by each machine are divided into two categories: random and deterministic. Each machine randomly interacts with its neighbors according to a probability distribution of its own; these are random interactions. Each machine reacts to the interactions it receives according to five simple rules; these are the deterministic interactions. In this network, each interaction is associated with four features. One of them is binary, and the other three are continuous, following a normal distribution. Each machine

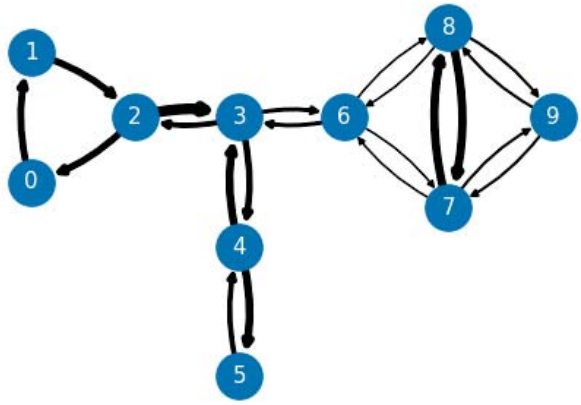

Fig. 2 Average behavior of the synthetic network

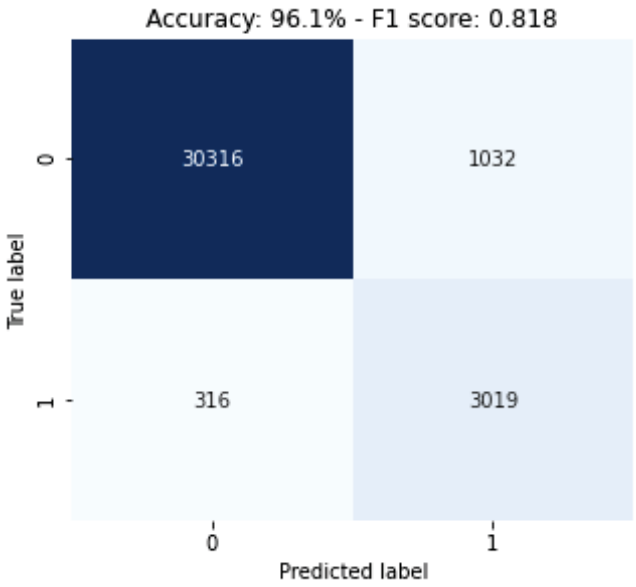

Fig. 3 Unsupervised anomaly detection results: Confusion matrix

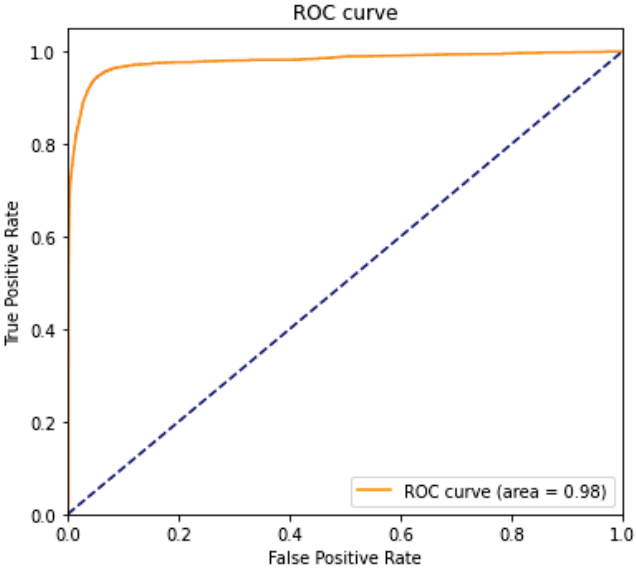

Fig. 4 Unsupervised anomaly detection results: ROC curve

has its parameters for the distribution governing the features associated with its interactions.

Generation of Anomalies: The next step after network construction and its normal interactions is to simulate abnormal interactions, which follow three scenarios:

- Scenario No. 1: Sending messages between two machines that do not normally interact with each other;
- Scenario No. 2: Sending a similar message from a machine to one of its neighbors several times in a short time;
- Scenario No. 3: Changing the parameters of the probability rule of a feature associated with a message.

Finally, we generate a training dataset without anomalies for the TGN and a test dataset including anomalies within normal data.

2) Evaluation: Our test dataset is not a table with normal and abnormal interactions independent of each other, allowing simple classifier evaluation. Our data are temporal and non-independent with abnormal periods rather than abnormal interactions. To get back to a normal classification evaluation, we divided the global period into small windows, each window considered as a test sample. In the following two cases, classification is done correctly: (1) the window contains an anomaly, and the anomaly indicator, A, exceeds the threshold at least once during the period; (2) the window has no anomaly, and A never exceeds the threshold. We chose time frame "windows" using the following procedure: we defined a fixed time frame, τ, and subdivided the global interval as $[t_i, t_{i+1}]$ in such a way that an anomaly always correspond to a date t_i, and $|t_{i+1} - t_i| = \tau$ except for cases in which t_{i+1} is the time of an anomaly, in that case $|t_{i+1} - t_i|$ may be shorter. Such "shorter" time frames are normalized in the evaluation procedure. This was a simple way of keeping almost constant time frames without overlapping anomalies in the "interval of observation."

3) Model Hyper-Parameters: In this section, you will find the hyper-parameters and the exact architecture that we used to obtain the results reported in Section V-B4. These were obtained by grid search optimizing the results.

- Message encoder: 1-layer MLP
- Message aggregator: Mean message
- Memory dimension: 100
- Memory updater: 1-layer GRU
- Embedding: 1-layer Temporal Graph Attention
- Decoder: 5-layers MLP
- False message generator: (1) Generate random source and destination, randomly sample features from training data ; (2) Sample real messages from the batch, randomly permute dummy features and multiply quantitative features by a random factor.
- Time encoding dimension: 10
- Node features: Count of communications with every other machines last 0.3, 1 and 10 seconds.

TABLE III
HYPERPARAMETERS FOR LEARNING AND FOR [INDICATOR]A

Initial learning rate	10^{-3}	Minimal learning rate	10^{-5}
Decay rate	$1 - 10^{-4}$	Batch size	10
β	0.25	α	0.25
ϵ	10^{-3}	Δt	0.02
threshold	0.284	-	-

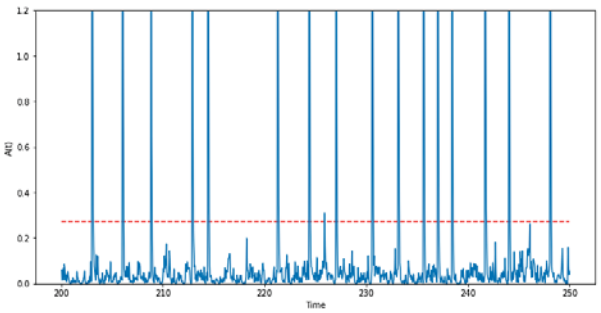

Fig. 5 Abnormality indicator A(t)

4) Results: Fig. 5 shows the evolution of the anomaly indicator $A(t)$ over 50 seconds. In red dashes is indicated the threshold. We can clearly distinguish spikes in the indicator, largely exceeding the threshold, which corresponds to abnormal messages. The indicator oscillates in a range close to zero for the rest of the time. Figs. 3 and 4 present the global results of the unsupervised approach. We obtain a global accuracy of 96.1%, an F1-score of 0.818, and an AUC ROC (area under the ROC curve) of 0.98, respectively. Table IV presents the accuracy according to each scenario of our test dataset. We can see that the method detects almost all messages with abnormal features (Scenario No. 3: 99.8% accuracy). The method also performs very well on messages between two machines that do not normally interact (Scenario No. 1: 92.7% accuracy). Finally, the method is less accurate when the anomaly increases the rate of messages between two machines (Scenario No. 2: 66.2% accuracy). However, it is useful to note that these figures depend on the choice of the threshold and can thus be improved at the cost of a decrease in accuracy in the normal scenarios, i.e., a higher false alarm rate.

TABLE IV
ACCURACY FOR EACH TYPE OF ANOMALY

Scenario No.	Normal	1	2	3
Accuracy	0.967	0.927	0.662	0.998

VI. DISCUSSION AND FUTURE WORKS

We proposed two systems for recognizing anomalies in a network in this work. The first one is based on supervised ML algorithms, and the second one is designed using a Temporal Graph Network. LR, DT, linear SVM, GBDT, and Random Forrest algorithms were implemented in the first system. By comparing the results of performance metrics, Table I, we concluded that LR is the best method for identifying anomalies in the proposed pipeline. According to Table I, the accuracy results obtained from the various algorithms are very close to

each other. After the removal of RF due to the lower accuracy of test data compared to training data, DT due to lower accuracy than other methods, and GBDT due to lower F1-score and FAR on test data compared to training data, among the other two methods, we chose the method that has the lower FAR. The FAR performance metric in this work is important because the existing data sets in cybersecurity are unbalanced. This imbalance stems from the fact that the number of regular interactions is much higher than the number of anomalous interactions in a network. Therefore, after accuracy, the "False Alarm Rate" is critical. We chose the LR method to build a trained model for the reasons mentioned. In the second phase of the supervised methods, we deployed our model to an online model so that we could analyze streaming data that are being sniffed from a network in real-time. The calculated performance metrics in the online mode show us that we had better results in the offline model. One of the essential factors in this reduction in performance can be the amount of exchanged data in the network, which is much lower than the amount of data in the training set. Another reason could be attack simulation because the method of simulating them in test data is different from that of simulating them in trained data. This method has been deliberately chosen differently so that we can test the efficiency of the algorithm in different modes. We compared the results obtained from the proposed method with other results in the literature. As can be seen, our results are better than many other results, but it is not the best result, and it is in third place by a very short margin. This difference is that we tried to use features that could be extracted in the online module, so we removed many features and did not use them for training the model. This removal can lead to a decrease in detection accuracy. However, since this decrease in accuracy is very small, those features can be ignored and instead reduce the time of extraction of features in accurate time detection. The second system in this paper is based on Temporal Graph Network. In this regard, we simulated two sets of data. The first set simulates a typical network behavior during a given period, while the second set simulates the traffic anomalies in the network according to the defined scenarios. The TGN is trained on the created data set using the modules defined in Section III-A1. Then we presented an indicator to distinguish abnormalities from normal behaviors based on a predetermined threshold. This system can detect anomalies with 96.7% accuracy. We will evaluate the proposed IDS systems in a network with real attacks and a greater variety of attacks in future work.

REFERENCES

[1] Godala, Sravanthi, and Rama Prasad V. Vaddella. "A study on intrusion detection system in wireless sensor networks." International Journal of Communication Networks and Information Security 12.1 (2020): 127-141.
[2] Lyon GF. Nmap network scanning: The official Nmap project guide to network discovery and security scanning. Insecure. Com LLC (US); 2008.
[3] Sarker, Iqbal H., et al. "Cybersecurity data science: an overview from machine learning perspective." Journal of Big data 7.1 (2020): 1-29.
[4] Moustafa N, Slay J. UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In2015 military communications and information systems conference (MilCIS) 2015 Nov 10 (pp. 1-6). IEEE.
[5] Seufert, Stefan, and Darragh O'Brien. "Machine learning for automatic defence against distributed denial of service attacks." 2007 IEEE International Conference on Communications. IEEE, 2007.
[6] Alazab, Ammar, et al. "Using feature selection for intrusion detection system." 2012 international symposium on communications and information technologies (ISCIT). IEEE, 2012.
[7] Buczak, Anna L., and Erhan Guven. "A survey of data mining and machine learning methods for cyber security intrusion detection." IEEE Communications surveys & tutorials 18.2 (2015): 1153-1176.
[8] Sarker, Iqbal H., A. S. M. Kayes, and Paul Watters. "Effectiveness analysis of machine learning classification models for predicting personalized context-aware smartphone usage." Journal of Big Data 6.1 (2019): 1-28.
[9] Li, Yinhui, et al. "An efficient intrusion detection system based on support vector machines and gradually feature removal method." Expert systems with applications 39.1 (2012): 424-430.
[10] Brugger, T. "KDD cup'99 dataset (network intrusion) considered harmful, 15 September 2007. Retrieved January 26, 2008." (2007).
[11] Hosseinzadeh, Mehdi, et al. "Improving security using SVM-based anomaly detection: issues and challenges." Soft Computing 25.4 (2021): 3195-3223.
[12] Yang, Kun, Samory Kpotufe, and Nick Feamster. "An Efficient One-Class SVM for Anomaly Detection in the Internet of Things." arXiv preprint arXiv:2104.11146 (2021).
[13] Shapoorifard, Hossein, and Pirooz Shamsinejad. "Intrusion detection using a novel hybrid method incorporating an improved KNN." Int. J. Comput. Appl 173.1 (2017): 5-9.
[14] Serpen, Gursel, and Ehsan Aghaei. "Host-based misuse intrusion detection using PCA feature extraction and kNN classification algorithms." Intelligent Data Analysis 22.5 (2018): 1101-1114.
[15] Salama, Mostafa A., et al. "Hybrid intelligent intrusion detection scheme." Soft computing in industrial applications. Springer, Berlin, Heidelberg, 2011. 293-303.
[16] Al-Yaseen, Wathiq Laftah, Zulaiha Ali Othman, and Mohd Zakree Ahmad Nazri. "Real-time intrusion detection system using multi-agent system." IAENG International Journal of Computer Science 43.1 (2016): 80-90.
[17] Yan, Weizhong, and Lijie Yu. "On accurate and reliable anomaly detection for gas turbine combustors: A deep learning approach." arXiv preprint arXiv:1908.09238 (2019).
[18] Protogerou, Aikaterini, et al. "A graph neural network method for distributed anomaly detection in IoT." Evolving Systems 12.1 (2021): 19-36.
[19] Kolias, Constantinos, et al. "DDoS in the IoT: Mirai and other botnets." Computer 50.7 (2017): 80-84.
[20] Rossi, Emanuele, et al. "Temporal graph networks for deep learning on dynamic graphs." arXiv preprint arXiv:2006.10637 (2020).
[21] Béres, Ferenc, et al. "Node embeddings in dynamic graphs." Applied Network Science 4.1 (2019): 1-25.
[22] Kazemi, Seyed Mehran, et al. "Representation Learning for Dynamic Graphs: A Survey." J. Mach. Learn. Res. 21.70 (2020): 1-73.
[23] Sokolova, Marina, and Guy Lapalme. "A systematic analysis of performance measures for classification tasks." Information processing & management 45.4 (2009): 427-437.
[24] DataReportal (2021), "Digital 2021 Global Digital Overview," retrieved from https://datareportal.com/reports/digital-2021-global-digital-overview
[25] Liu, Hongyu, and Bo Lang. "Machine learning and deep learning methods for intrusion detection systems: A survey." applied sciences 9.20 (2019): 4396.