A Neural Attention Model for Real-Time Network Intrusion Detection

Mengxuan Tan,* Alfonso Iacovazzi,* Ngai-Man (Man) Cheung,* Yuval Elovici*†
*ST Engineering-SUTD Cyber Security Laboratory – Singapore University of Technology and Design, Singapore
†Department of Software and Information Systems Engineering – Ben-Gurion University of the Negev, Beer-Sheva, Israel
Email: {mengxuan_tan2, alfonso_iacovazzi, ngaiman_cheung, yuval_elovici}@sutd.edu.sg

***Abstract*—The diversity and ever-evolving nature of network intrusion attacks has made defense a real challenge for security practitioners. Recent research in the domain of Network-based Intrusion Detection System has mainly focused on adopting a flow-based approach when extracting features from raw packets. One drawback of this is that attack detection can only be carried out after the flow has ended. In this work, we present a new technique based on the neural attention mechanism; unlike many existing solutions, our technique can be applied for real-time attack detection since it uses time slot-based features. The proposed solution is a modified version of the transformer model which has been proposed and used in the language translation domain. We conduct experiments on a dataset extracted from a recent repository network traffic containing several kinds of network attack. We use the "bidirectional LSTM" and "conditional random fields" models as baseline for comparison and our performance results demonstrate that the proposed solution significantly outperforms the two baselines in terms of precision, recall, and false positive rates. In addition, we show that our solution is more computationally efficient than the bidirectional LSTM model as a result of the removal of recurrent layers.**

***Index Terms*—Network intrusion detection, Deep learning, Attention model, Network security.**

I. Introduction

Network Intrusion Detection Systems (NIDSs) are well-known and widely used systems aiming at analyzing the network traffic so as to identify traffic anomalies and/or cyber attacks against the networked devices. For this reason, NIDSs play a fundamental role in the process of protection of enterprise networks.

Traditional NIDS solutions can be categorized in two classes: anomaly-based and signature-based NIDSs. The former solutions check whether the network flows correctly follow the protocol specifications and do not show any misuses or improper uses of the network; this usually relies on the knowledge of the normal and correct behavior of the network traffic. On the contrary, the signature-based NIDSs deep inspect the network flow packets and look for specific patterns which are known to be distinctive signatures of some cyber attacks; this requires to have a deep knowledge of the network attacks so as to identify the signatures characterizing the malicious traffic. However, this last solutions are not able to detect zero-day attacks, since their behavior is unknown to developers.

Unfortunately, traditional NIDSs are becoming obsolete due to three main reasons: (i) the significant increasing of the network traffic volume which made deep packet inspection challenging, (ii) the evolution of the cyber attacks which are becoming more and more advanced and able to evade NIDSs, (iii) the widespread use of encryption which does not allow the deep inspection of packet contents.

To overcome these limitations, statistical analysis approaches have been investigated to be utilized in this domain. Traditional machine learning techniques (e.g., Support Vector Machine, Nave Bayes, etc.) have been tested with promising results in order to detect network traffic attacks by analyzing only statistical features. Recently, new approaches based on deep learning models have been proposed with higher detection rates [1]. However, these approaches have several limitations. First, although the detection rates can be quite high, these models suffer from high false alarm rates, which is not acceptable for a network anomaly detection since each alarm triggers actions for further investigation [2], [3], [4]. Additionally, since the main datasets available in this domain include only flow-based features, which can be computed only at the end of a flow (e.g., flow duration), the state-of-the-art machine learning solutions cannot detect an attack in real time, but only when the attack is over. This is a main concern since a network administrator needs to be warned as soon as possible about an attack in order to mitigate its effects. Finally, most of the solutions proposed in the literature use out of date datasets (e.g., KDD 1999 [5]), which do not contain recent and more sophisticated network attacks.

In contrast with the majority of solutions in the literature which use flow-based features, the solution proposed in this paper takes time slot-based traffic features as input; this facilitates real-time attack detection. Although these features have a sequence structure, we do not implement a Recurrent Neural Network to model this structure. Instead, starting from the findings and results obtained by Vaswani in 2017 [6], we apply the "attention" mechanism, which was proposed for the language translation task, to the sequence modeling problem in intrusion detection. Our model is an adaptation of Vaswani's transformer model to the intrusion detection task. Although we use "attention" and "self-attention" components like those in the transformer model, the structure of our model is much simpler since it is not built on an Encoder-Decoder architecture which is a typical format for language translation.

The main contributions of this paper are:

- the proposal of a simple and effective neural model based on the attention mechanism for detecting attacks in the network traffic;

- the use of only pure time slot-based traffic features as input of a NIDS, which enables and facilitates real-time detection;
- a wide analysis of the proposed solution's performance which is compared to that obtained by two baseline models (bidirectional LSTM and conditional random fields).

The remainder of the paper is organized as follows. An overview of related works in network intrusion detection is provided in Section II. Section III provides a description of the traffic features used and their pre-processing. The architecture of the proposed solution is described in Section IV. The performance evaluation is provided in Section V, and finally Section VI presents our conclusions.

II. Related work

Network intrusion detection has been widely investigated during the last two decades. Many surveys have been written due to the importance of this topic [7], [8], [9], [10], [11]. One of the first important survey in this domain is provided by Mukherjee *et al.* in 1994 [7]. Since then a huge number of approaches for network intrusion detection have been proposed to address the many issues in this domain.

Deep learning based NIDSs have gained a lot of attention, because these new machine learning algorithms seem to be much more powerful than traditional approaches [11]. The two recent surveys by Kwon *et al.* and Hodo *et al.* provide an overview and comparison of network intrusion detection solutions which are based on deep learning mechanisms [1], [12]. In this sub-domain, the research community has proposed a number of NIDSs which are based on: (i) Convolutional Neural Networks (CNNs) [13], (ii) Deep Autoencoders (DAs) [14], (iii) Deep Belief Networks (DBNs) [15] and (iv) Recurring Neural Networks (RNNs) [16]. In general, it has been proved that deep neural networks significantly outperforms the traditional machine learning algorithms (e.g., random forest, naive bayes, etc.). Nonetheless, the class of RNN algorithms, which includes Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU) showed better performance compared to other neural network families, due to their capacity to capture the sequential information in the network traffic [17], [18]. However, weaknesses of the RNN such as the problem of vanishing or exploding gradient, difficulty in learning long-term dependencies, and poor sequential computation efficiency [19], [20], may hinder the detection performance of these models.

A recent trending topic in deep learning is the use of attention mechanisms to be incorporated in traditional models to improve sequence modeling [21], [22], [23]. For this reason, some researchers have tried to apply attention mechanisms to the intrusion detection field. For instance, Brown*et al.* [24] incorporated the attention mechanism in an RNN language model in order to detect anomalies in system log lines. Zhu *et al.* [25] proposed the Attention-based Multi-Flow LSTM (AMF - LSTM) to identify abnormal traffic flows during intrusion attacks. Attention mechanism is also used in Qin *et al.* [26] to train RNNs in order to identify segments of a payload script the model should place attention on when deciding if the payload is malicious; unfortunately, this solution cannot be applied when the packet payloads are encrypted.

One important limitation of the mentioned deep learning approaches for network traffic analysis - be they convolutional, recurrent, or attention-based - is that they use flow-based features as input to their models, hence detection can only be carried out after the flow has ended. In other words, preventive measures can only be taken after the attack has ended.

III. Traffic features and pre-processing

The majority research works in network intrusion detection try to detect network attacks by using flow-based features. Taking a clue from the research of Casas *et al.* [27], a network flow in our intrusion detection solution is not handled as single sample of data, but we consider the aggregation of all network flows observed in a network collection point. The time axis is slotted in time slots with fixed duration δ and a number of statistical features are defined over the packets observed during a single time slot. The resulting dataset is a sequence of ordered data units describing the traffic over the time. More specifically, we selected $d = 55$ time slot-based features which are listed in Table I.

Most of the features can be easily computed from the only network packets observed during the time slot. However, the last four features in Table I require information from the previous time slots, to be computed. In detail, to compute the number of flows active in a time slot, it is required to keep memory of all the flows which have not been terminated. A flow is considered active for a given time slot starting at the time t_s, if no FIN packet of that flow has been observed before the time t_s (only for TCP flows) and the registered time t_{last} of the last packet observed in that flow is within a short interval: $t_{last} > t_s - \tau$ (with $\tau = 10\ sec$).

The dataset has been normalized by using the StandardScaler method from the sklearn library [28] which to standardize the features by subtracting the mean of each feature from the data samples before dividing it by the standard deviation.

IV. Model architecture

Our model, which we call Attention for Network Intrusion Detection (ANID) model, draws inspiration from the transformer model [6] which was introduced by Vaswani in 2017 for the language translation task. This model achieved great performance in English-to-German and English-to-French translation tasks on the standard WMT 2014 English-to-German and English-to-French datasets. The architecture of this model relies solely on attention mechanisms to carry out the language translation task. In spite of its simplicity, the transformer model outperforms previous state-of-the-art models in terms of accuracy and computation speed without the use of any recurrent or convolution layer.

In order to utilize the attention mechanism in the network intrusion detection domain and take advantage of its properties,

TABLE I
LIST OF FEATURES

Feature names	Description
nPktsUp, nBytesUp, nProtsUp, nAddrsUp, nPortsUp,nPktsDo, nBytesDo, nProtsDo, nAddrsDo, nPortsDo, nPktsBi, nBytesBi, nProtsBi, nAddrsBi, nPortsBi	number of packets, bytes, protocols, unique IP addresss, and unique ports counted in upstream, downstream, and both directions.
n(XXX)sUp, n(XXX)sDo, n(XXX)sBi	number of TCP packets with flag XXX set (1), with XXX in {FIN, SYN, RST, PSH, ACK, URG, ECE, CWR}, counted in upstream, downstream, and both directions.
avgPktLenUp, stdPktLenUp, 25thPktLenUp, 75thPktLenUp, avgPktLenDo, stdPktLenDo, 25thPktLenDo, 75thPktLenDo, avgPktLenBi, stdPktLenBi, 25thPktLenBi, 75thPktLenBi	average value, standard deviation, 25-th and 75-th percentile of the packet lengths in upstream, downstream, and both directions.
nActFlowsUp, nActFlowsDo	number of flows active in the time slot, in upstream and downstream, respectively.
avgFlowDur, stdFlowDur	average value and standard deviation of the duration of the active flows up to the current time slot.

we designed a simplified version of the transformer model. Our model is depicted in Figure 1.

The main difference between Vaswani's and our model is that the former has an encoder-decoder structure, since it is used for the neural machine translation task, while the latter uses a simple feed-forward network with no loop involved in the connections between units. In addition, we use a weighted loss function in ANID model to deal with the disproportionate ratio of samples from the classes in our dataset.

The ANID model is composed by three main components: the Feed Forward (FF) module, the Scaled Dot-Product Attention (SDA) module, and the Scaled Dot-Product Self-Attention (SDSA) module. These modules are the same used in the transformer model, though differently combined, and are described in detail in the next subsections. The model takes as input a sequence of T time-slot data units. Then, the input is linearly transformed by the Input Embedding and Add modules. The resulting data traverses the FF, SDA, SDSA modules, and a final output layer. The final output of the model is made of T predictions, one for each input time slot.

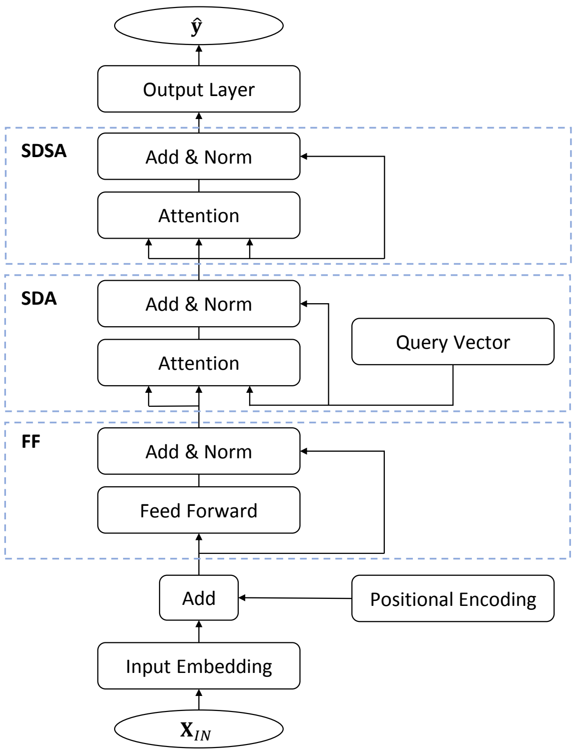

Fig. 1. Block diagram of the ANID model.

A. Input Embedding and Positional Encoding

The first component of the ANID model is the "Input Embedding" module. Given the raw input sample $\mathbf{X}_{IN} \in \mathbb{R}^{T\times d}$, in which each row corresponds to a feature vector in a time slot, the Input Embedding enforces a linear transformation to the raw input according to equation:

$$\mathbf{X}_{IE} = (\mathbf{X}_{IN} \cdot \mathbf{W}_0 + \mathbf{b}_0) + \mathbf{P} \tag{1}$$

in which $\mathbf{W}_0 \in \mathbb{R}^{d\times h}$ is the linear transformation matrix shared across all the time slots, $\mathbf{b}_0 \in \mathbb{R}^h$ is the bias, h is the hidden size, and $\mathbf{P} \in \mathbb{R}^{T\times h}$ is the matrix of positional embeddings [29]. $\mathbf{P}$ and $\mathbf{W}$ are randomly initialized and updated with the other parameters of the model during the training phase. Since recurrent layers commonly found in RNNs are not present in the ANID model to learn about the order of time slot data units within a sample, these positional embeddings are used to incorporate relative positional information about the time slot units in a sequence. According to the notation used by Vaswani in [6], the positional embeddings matrix is represented in Figure 1 as embedded by the module "Positional Encoding."

B. Feed Forward block

The output $\mathbf{X}_{IE} \in \mathbb{R}^{T\times h}$ obtained from the linear transformation is not directly sent to the attention modules, but is first processed by a fully connected Feed Forward (FF) network, and a subsequent layer normalization [30] with residual connection [31]. The feed-forward block is the same used by Vaswani in [6], in which two linear projections are enforced with a ReLU activation in between:

$$\mathrm{FFN}(\mathbf{X}_{IE}) = \mathrm{ReLU}(\mathbf{X}_{IE} \cdot \mathbf{W}_1 + \mathbf{b}_1) \cdot \mathbf{W}_2 + \mathbf{b}_2 \quad (2)$$

$$\mathbf{X}_{FF} = \mathrm{Norm}(\mathbf{X}_{IE} + \mathrm{FFN}(\mathbf{X}_{IE})) \quad (3)$$

The variables $\mathbf{W}_1 \in \mathbb{R}^{h \times p}$, $\mathbf{W}_2 \in \mathbb{R}^{p \times h}$, $\mathbf{b}_1 \in \mathbb{R}^{p}$, and $\mathbf{b}_2 \in \mathbb{R}^{h}$ in Equation 2 are shared parameters across inputs from different positions. The output of Equation 3 is finally sent to the SDA module.

C. Attention module

The "attention" module is the core of both SDA and SDSA components in our model. This module receives three matrices as input: a matrix of "keys" $\mathbf{K} \in \mathbb{R}^{T \times h}$, a matrix of "values" $\mathbf{V} \in \mathbb{R}^{T \times h}$, and a matrix of "queries" $\mathbf{Q} \in \mathbb{R}^{T \times h}$. Its output is a context matrix $\mathbf{C} \in \mathbb{R}^{T \times h}$ which is computed as:

$$\mathbf{C} = \mathrm{Attention}(\mathbf{K}, \mathbf{Q}, \mathbf{V}) = \mathrm{Softmax}\left(\frac{\mathbf{Q} \cdot \mathbf{K}^T}{\sqrt{h}}\right) \cdot \mathbf{V} \quad (4)$$

The (i, j)-th element of the matrix resulting from the dot-product $\mathbf{Q} \cdot \mathbf{K}^T$ gives a measure of the pairwise similarity between the i-th row of $\mathbf{Q}$ and the j-th row of $\mathbf{K}$, which represents the attention score given to the (i, j)-th (query,key) pair. Therefore, the context matrix contains information about the intrinsic dependencies between latent representations of a sequence of time slot data units $\mathbf{X}$ within a data sample. The context vector is then used in downstream tasks to assist in the predictions of $\hat{\mathbf{y}}$.

In the SDA component, the three inputs sent to the Attention module are obtained as follow: the two matrices of "keys" and "values" are both equal to the output $\mathbf{X}_{FF}$ coming from the FF module ($\mathbf{K} = \mathbf{V} = \mathbf{X}_{FF}$); similar to the positional embeddings $\mathbf{P}$, the matrix of "queries" $\mathbf{Q}$ are also trainable parameters that are being randomly initialized and learned during the training phase. Additionally, a residual connection is applied around the attention mechanism, followed by a layer normalization:

$$\mathbf{X}_{SDA} = \mathrm{Norm}(\mathbf{Q} + \mathrm{Attention}(\mathbf{X}_{FF}, \mathbf{X}_{FF}, \mathbf{Q})) \quad (5)$$

The output after the layer normalization is then passed to the next module.

The structure of the SDSA component is similar to that of SDA, with the only difference that in the SDSA the three inputs sent to the Attention module are identical (Equation 6). The goal of this type of module is to capture deep contextual information within a sequence of data by using self-attention [6].

$$\mathbf{X}_{SDSA} = \mathrm{Norm}(\mathbf{X}_{SDA} + \mathrm{Attention}(\mathbf{X}_{SDA}, \mathbf{X}_{SDA}, \mathbf{X}_{SDA})) \quad (6)$$

D. Output layer

The output $\mathbf{X}_{SDSA}$ given by the SDSA module passes through a final fully connected layer with softmax activation function which outputs the predictions $\hat{\mathbf{y}}$ of the time slot labels.

V. Performance analysis

We carry out a set of experimental evaluations in order to provide a deep analysis of the performance of the proposed model. Then, we compare the overall results against baseline models. The models are tested on a dataset which has been extracted from the CICIDS2017 repository: a public and recent network traffic collection provided for network intrusion detection purposes.

A. Dataset description

The CICIDS2017 [32] is a publicly available repository consisting of raw pcap network traces collected in a controlled framework over a period of five days. The traces contain traffic carrying seven different types of attack overlapping with normal/genuine traffic. The attacks are: Denial of Service (DoS), Distributed Denial of Service (DDoS), Infiltration, Brute Force, Port Scan, Web attack, and Botnet.

Starting from the raw pcap traces, we generated a time slot-based dataset, according to the features described in Section III. Given the time t_0 of the first packet in a trace, and the time slot width $\delta = 0.5\ sec$, the i-th time slot is the interval $[i \cdot t_0, i \cdot t_0 + \delta)$. A feature vector $\mathbf{x}_i$ is associated to the i-th time slot. Also, the i-th time slot is labeled with a binary value y_i: $y_i = 0$ if there is no attack within the time slot (normal class), otherwise $y_i = 1$ (attack class). The ground truth is provided along with the traffic repository CICIDS2017.

The number of slots under the attack class is approximately 20% of the total number of slots in the entire dataset. In order to simulate an actual cyber system environment where malicious traffic is rarely seen, we further reduced the proportion of time slots in the attack class from 20% to approximately 1% by removing approximately the last 95% of the attack time slots in time order from all types of attacks.

Since the model takes as input a sequence of T data units, we split the whole sequential dataset in samples made of T time slot feature vectors, this is obtained by sliding over the sequential dataset with step 1 and window T. Then, the j-th sample of our dataset is given by $S_j = \{(\mathbf{x}_i, y_i) | i = j, ..., j + T - 1\}$. We selected $T = 10$ in our experiments.

B. Feature set reduction

To reduce the size of the input data, we use a univariate feature selection algorithm based on the Pearson correlation coefficient (PCC). We compute the absolute PCC scores between each feature and the label; the absolute PCC scores are used as ranks of the features and the top features whose scores are greater or equal to a threshold of 0.08 are selected. Table II shows the selected features and their corresponding PCC values.

TABLE II
FEATURE'S PCC SCORES

Features	PCC Score	Features	PCC Score
nSYNsUp	0.15	nPortsUp	0.14
nSYNsBi	0.14	nRSTsBi	0.14
nRSTsDo	0.14	nPortsBi	0.13
nActFlowsUp	0.11	nActFlowsDo	0.10
nPktsUp	0.10	nPktsBi	0.09
nAddrsDo	0.09	nACKsDo	0.09
nPktsDo	0.09	nACKsBi	0.09
nACKsUp	0.08	nBytesUp	0.08
nBytesBi	0.08	nBytesDo	0.08
stdPktLenBi	0.08		

C. Baseline: bidirectional long short-term memory

The long short-term memory (LSTM) model [33] is a popular type of recurrent neural network (RNN) which is widely used for sequence data modeling due to its key advantage in overcoming long-term dependency problem. Like all RNN models, LSTM has a sequential structure made of loops over a basic LSTM cell. Given a generic sequence of T data units in the dataset, the input of the i-th LSTM step is given by (i) the i-th data unit of the sequence and (ii) the output states coming from the previous steps:

$$(\mathbf{h}_i, \mathbf{c}_i) = \text{LSTMCell}(\mathbf{h}_{i-1}, \mathbf{c}_{i-1}, \mathbf{x}_i) \tag{7}$$

The parameters $\mathbf{h}_i$ and $\mathbf{c}_i$ in Equation 7 are the hidden state and cell state of the i-th LSTM step. While the hidden state is used to convey the information processed by short-distance cell state, the cell state is used to propagate the long-term information. We do not provide the detailed structure of the LSTM cell; the reader may refer to [33] for more details.

A final softmax layer takes the hidden states $\mathbf{h}_i$ as input and outputs the model's prediction $\hat{y}_i$.

The Bi-LSTM model is the bidirectional version of LSTM, and is able to process data sequences in both the forward and backward directions by incorporating two LSTM cells.

We use the Bi-LSTM network as baseline model because it usually obtains high performance in sequence modelling problems and it is widely used in many domains as benchmarking reference.

D. Baseline: conditional random fields

We also use a traditional machine learning model that is commonly used for sequence modelling as an additional baseline for comparison. The linear-chain conditional random fields (CRF) [34] is a probabilistic model that learns the conditional probability of $Pr(\mathbf{y}|\mathbf{X}_{IN})$.

CRF has been widely used in many domains and has achieved decent performance for tasks such as image labeling [35] and name entity recognition [36].

E. Dataset splitting

The full main dataset was split in three sub-datasets: training, validation, and test dataset. We shuffled all the samples and split them according to the following proportion: 60% of the samples in the training set, 20% of the samples in the validation set, and 20% of the samples in the test set. The original proportions between attacks and normal time slots have been preserved in the three datasets. The sample shuffling done before splitting allows all three datasets to contain samples from diverse time ranges.

F. Loss function

In order to deal with the imbalanced samples in our dataset, we use the weighted cross-entropy loss function for training the Bi-LSTM and ANID models which is computed as

$$Loss = -\sum_{j=1}^{N}\sum_{i=1}^{T} \beta \cdot y_{i,j} \cdot \log(\hat{y}_{i,j}) + (1-y_{i,j}) \cdot \log(1-\hat{y}_{i,j}) \tag{8}$$

in which $y_{i,j}$ is the i-th binary label of the j-th sample in the training dataset, N is the total number of data samples in the training set, and β is the ratio between the number of time slots with no attack and the number of time slots with an attack.

On the other hand, CRF uses the negative log-likelihood function [34] to estimate its model parameters:

$$Loss = -\sum_{j=1}^{N} \log Pr(\mathbf{y}_j|\mathbf{X}_{INj}). \tag{9}$$

G. Regularization

We use the dropout [37] as a method of regularization to prevent both the Bi-LSTM model and ANID model from overfitting during the training phase. In the Bi-LSTM model, dropout is being applied to both the input and output of the Bi-LSTM at every time slot. On the other hand, we apply dropout to the output $\mathbf{X}_{IE}$ of Equation 1 for the ANID model. A dropout rate of 0.1 is being used for both models.

We do not apply regularization to the CRF model given the small amount of parameters (Table IV) it has and its inability to output high F-Score on the training set (Figure 2).

H. Evaluation metrics

All three models output a sequence of predictions $\hat{\mathbf{y}}$ for every input sample $\mathbf{X}$ in a dataset. Given an instance of classification, let TP, FP, TN, and FN be the number of true positives, false positives, true negatives, and false negatives obtained by the classification; we use the predictions $\hat{\mathbf{y}}$ and the target labels $\mathbf{y}$ to compute the following performance metrics:

- **Precision**: The proportion of the time slots predicted as attack that are truly attack:

$$PREC = \frac{TP}{TP + FP} \tag{10}$$

- **Recall**: The rate at which the true attack slots are accurately predicted as attacks:

$$REC = \frac{TP}{TP + FN} \tag{11}$$

- **False Positive Rate (FPR)**: The proportion of time slots predicted as attack that are not attack:

$$FPR = \frac{FP}{FP + TN} \tag{12}$$

- **F-score**: The harmonic mean of the precision and recall:

$$F = 2 \cdot \frac{PREC \cdot REC}{PREC + REC} \tag{13}$$

We focus our analysis on the last two metrics because the F-score is a good indicator of how well the models perform on imbalanced datasets, and the FPR is widely used in cyber security, since false alarms may negatively impact on the security management.

I. System specifications

We train all three models on a workstation with a 8x core E5-1620 v3 CPU @ 3.50GHz, with 32GB RAM and one NVidia GeForce GTX 1080 GPU. Throughout the experiments, we use a minibatch size of 256 samples and the Adam Optimizer [38] at a learning rate of 0.001 as the optimization method to perform stochastic gradient descent on the models during the training phase. The models' architectures are built using Tensorflow [39] v1.8.0 in Python v3.5.2.

J. Selecting hyperparameters

We carried out hyperparameters tuning for both the Bi-LSTM and ANID model during the validation phase, and we selected those that allow the models to give the highest F-score on the validation dataset. We fixed 600 hidden units for the Bi-LSTM model, and have set $h = 100$, $p = 800$ for the ANID model (with h be the size of $\mathbf{W}_0$ and $\mathbf{P}$, and p be the size of $\mathbf{W}_1$ and $\mathbf{W}_2$).

Unlike deep learning models, the parameters of CRF are fixed by the dimensions of $\mathbf{X_{IN}}$ and $\mathbf{y}$ [34]. Hence hyperparameters tuning is not carried out for CRF.

K. Results

During the training phase, the models are trained on the training dataset and evaluated on both training and validation datasets simultaneously. Figure 2 shows the loss and the F-score computed on the two datasets as function of the number of epochs for all of the models. To prevent models from excessive training, we used early stopping [40] by monitoring the F-score on the validation dataset after each epoch.

After multiple rounds of training, model training was early stopped at epoch 1500, 800, and 3000 for the Bi-LSTM, CRF, and ANID model, respectively. The model instances obtaining the highest F-score with the validation dataset, are used to evaluate the performance on the test dataset.

The overall results obtained on the test dataset for the Bi-LSTM, CRF, and ANID models are listed in Table III. All the considered metrics show that the performance of the ANID model outperforms that obtained by the Bi-LSTM and CRF models. In particular, we are able to significantly increase the recall and precision, while keeping the FPR very low.

TABLE III
MODELS' PERFORMANCE

Models	**Precision (%)**	**Recall (%)**	**F-Score (%)**	**FPR (%)**
Bi-LSTM	88.86	91.85	90.33	0.15
CRF	48.09	15.41	23.34	0.22
ANID	97.29	94.40	95.28	0.03

In order to provide an evaluation of the complexity of the three analyzed models, we list in Table IV the number of parameters required by each model, the computational speed in the training phase measured as the average time required per epoch, the convergence time, and the time required for a single test. We notice that the number of training parameters required by the Bi-LSTM is almost five times higher than those required by the ANID. On the other hand, despite the second column of Table IV shows that a single training epoch of ANID lasts less than that of Bi-LSTM, the ANID is slower than Bi-LSTM because ANID's training convergence requires about 1500 epochs more than Bi-LSTM. Nonetheless, the time required for the prediction by ANID is much lower than the time required by Bi-LSTM. Table V shows how the parameters are distributed among the several components in the ANID model.

Although CRF has the least number of parameters and fastest training and convergence time, it fails to perform in all four evaluation metrics (Table III).

TABLE IV
COMPLEXITY ANALYSIS

Models	**No. of params**	**Training time (sec/epoch)**	**Convergence time (sec)**	**Testing time (sec/sample)**
Bi-LSTM	769,202	22	33,000	0.37
CRF	48	11	8,800	0.32
ANID	165,702	17	51,000	0.27

TABLE V
NO. OF PARAMETERS FOR THE COMPONENTS IN ANID MODEL

Components	**No. of parameters**
IE & PE	3,000
FF	161,100
SDA	1,200
SDSA	200
Output	202

Finally, we evaluate the performance of the three considered models on each attack class separately. The results are shown in Table VI, VII and VIII, respectively. The results highlight that ANID outperforms the Bi-LSTM and CRF models for each of the attacks in our dataset. Additionally, we notice that results are quite homogeneous throughout the attack classes.

L. Combination study

We conducted a set of combination experiments in order to investigate the relevance and effectiveness of each module in the ANID model. Table IX shows the performance obtained in several combinations of the main four modules of ANID.

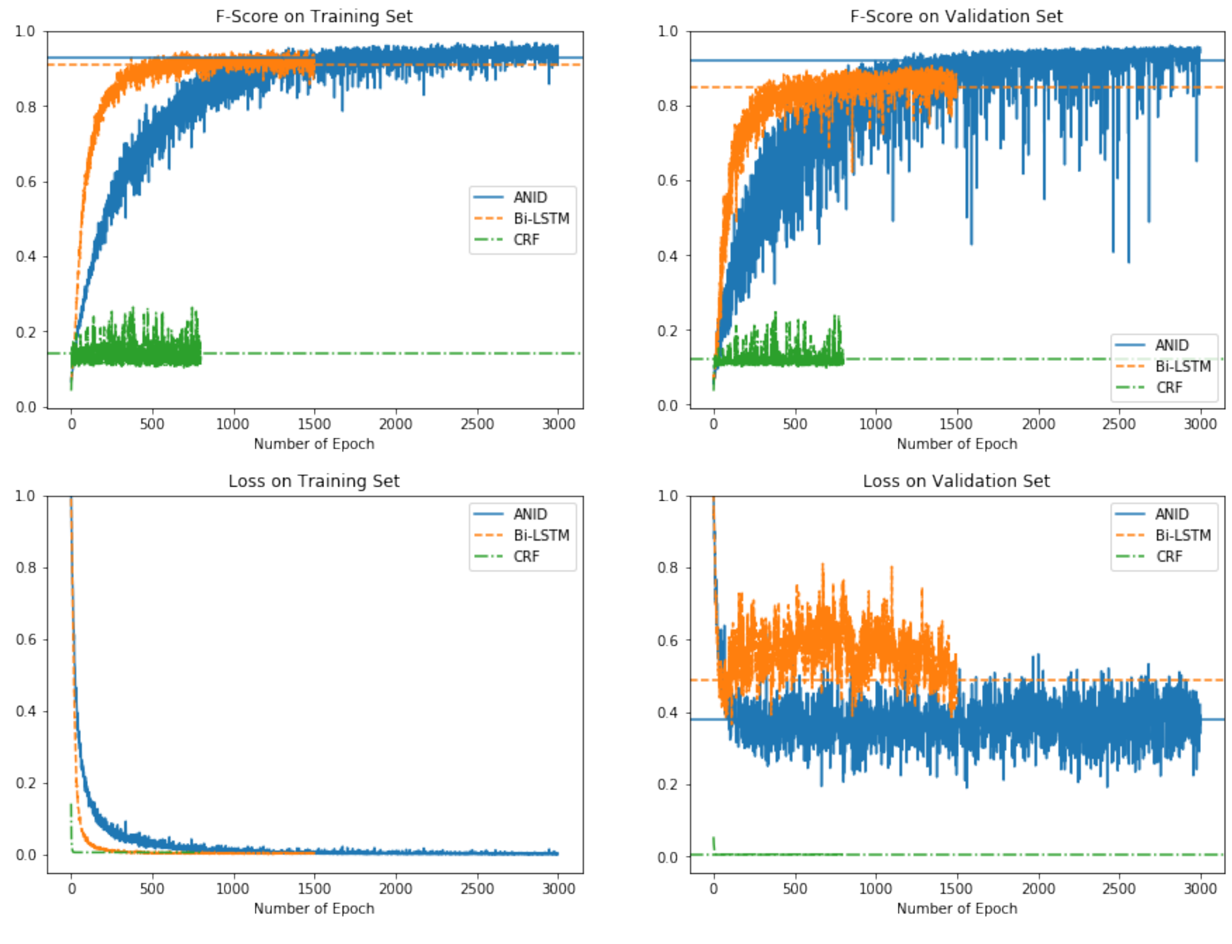

Fig. 2. Comparison of the loss and F-score convergence between the ANID , Bi-LSTM and CRF models on the training and validation datasets, respectively.

TABLE VI
CRF'S PERFORMANCE ON EACH ATTACK

Attacks	Precision (%)	Recall (%)	F-Score (%)
DoS	14.70	2.87	4.80
DDoS	52.54	18.43	27.29
Infiltration	32.32	7.95	12.76
Brute Force	70.89	40.53	51.57
Port Scan	60.83	25.86	36.29
Web attack	0.00	0.00	0.00
Botnet	0.00	0.00	0.00

TABLE VII
BI-LSTM'S PERFORMANCE ON EACH ATTACK

Models	Precision (%)	Recall (%)	F-Score (%)
DoS	88.64	89.99	89.31
DDoS	87.00	77.18	81.79
Infiltration	89.26	95.85	92.44
Brute Force	89.41	97.37	93.22
Port Scan	88.35	87.44	87.89
Web attack	88.11	85.44	86.76
Botnet	89.48	98.13	93.61

It is clear that when the model consists of only one or two modules the results are very poor. Combining three different modules allows significant improvement in performance, in particular we obtain very good performance with the combination "PE+FF+SDA" reaching above 90% precision and 0.05% FPR. Inverting the two modules FF and SDA does not lead to the same high performance. Adding the "SDSA" module to "PE+FF+SDA" combination allows to additionally improve the results to 97% precision and 0.03% FPR. This is also the best performance among all of the combinations of the four modules.

VI. CONCLUSIONS

In this work, we presented the ANID model, a new neural model based on the attention mechanism which can be used for real-time network intrusion detection. Real time detection is guaranteed by the use of time slot-based features extracted from network traffic. We tested the model against a recent network traffic trace containing several classes of attacks. Our results show that this model is able to (i) accurately and efficiently detect the attacks available in our dataset, and (ii) outperform the Bi-LSTM and CRF models, which we used as our baselines. In the future, we aim to incorporate attention mechanisms in unsupervised learning models that can be used

TABLE VIII
ANID'S PERFORMANCE ON EACH ATTACK

Attacks	Precision (%)	Recall (%)	F-Score (%)
DoS	97.27	94.09	95.66
DDoS	96.90	82.39	89.06
Infiltration	97.38	98.09	97.74
Brute Force	97.38	98.10	97.74
Port Scan	97.28	94.23	95.73
Web attack	97.01	85.63	90.97
Botnet	97.41	99.15	98.27

TABLE IX
COMBINATION STUDY

Components	Precision (%)	Recall (%)	F-Score (%)	FPR (%)
PE	3.24	54.81	6.12	21.18
PE+SDSA	3.71	66.23	7.02	22.24
PE+FF	47.16	96.37	63.33	1.40
PE+SDA	6.31	62.64	11.47	12.02
PE+SDSA+FF	55.95	93.39	69.97	0.95
PE+SDSA+SDA	5.92	78.70	11.01	16.18
PE+FF+SDSA	86.62	93.46	89.91	0.19
PE+FF+SDA	96.37	95.56	95.96	0.05
PE+SDA+FF	80.31	85.81	82.97	0.27
PE+SDA+SDSA	7.54	70.90	13.63	11.24
PE+SDSA+SDA+FF	79.30	79.92	79.61	0.27
PE+FF+SDSA+SDA	95.99	95.43	95.71	0.05
PE+FF+SDA+SDSA	97.29	94.40	95.26	0.03
PE+SDA+FF+SDSA	80.28	84.49	82.33	0.27
PE+SDA+SDSA+FF	77.30	85.90	81.37	0.33
PE+SDSA+FF+SDA	95.43	90.92	93.12	0.06
PE+FF+SDA+SDA	96.83	93.34	95.06	0.04
PE+FF+SDA+FF	95.79	94.57	95.18	0.05

for the detection of zero-day attacks.

ACKNOWLEDGEMENT

This work was supported by both ST Electronics and the National Research Foundation (NRF), Prime Ministers Office, Singapore under Corporate Laboratory @ University Scheme (Programme Title: STEE Infosec - SUTD Corporate Laboratory).

REFERENCES

[1] D. Kwon, H. Kim, J. Kim, S. C. Suh, I. Kim, and K. J. Kim, "A survey of deep learning-based network anomaly detection," *Cluster Computing*, pp. 1–13, 2017.

[2] M. S. Hoque, M. Mukit, M. Bikas, A. Naser *et al.*, "An implementation of intrusion detection system using genetic algorithm," *arXiv preprint arXiv:1204.1336*, 2012.

[3] B. Morin, L. Mé, H. Debar, and M. Ducassé, "M2d2: A formal data model for ids alert correlation," in *RAID*. Springer, 2002.

[4] T. Pietraszek, "Using adaptive alert classification to reduce false positives in intrusion detection," in *RAID*. Springer, 2004.

[5] "The kdd cup 1999 dataset," http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[6] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017.

[7] B. Mukherjee, L. T. Heberlein, and K. N. Levitt, "Network intrusion detection," *IEEE network*, vol. 8, no. 3, pp. 26–41, 1994.

[8] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *computers & security*, vol. 28, no. 1-2, pp. 18–28, 2009.

[9] I. Butun, S. D. Morgera, and R. Sankar, "A survey of intrusion detection systems in wireless sensor networks," *IEEE communications surveys & tutorials*, vol. 16, no. 1, pp. 266–282, 2014.

[10] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *IEEE SP*, 2010.

[11] R. Chapaneri and S. Shah, "A comprehensive survey of machine learning-based network intrusion detection," in *Smart Intelligent Computing and Applications*. Springer, 2019, pp. 345–356.

[12] E. Hodo, X. Bellekens, A. Hamilton, C. Tachtatzis, and R. Atkinson, "Shallow and deep networks intrusion detection system: A taxonomy and survey," *arXiv preprint arXiv:1701.02145*, 2017.

[13] R. Vinayakumar, K. Soman, and P. Poornachandran, "Applying convolutional neural network for network intrusion detection," in *IEEE ICACCI*, 2017.

[14] A. Javaid, Q. Niyaz, W. Sun, and M. Alam, "A deep learning approach for network intrusion detection system," in *BIONETICS*, 2016.

[15] N. Gao, L. Gao, Q. Gao, and H. Wang, "An intrusion detection model based on deep belief networks," in *IEEE CBD*, 2014.

[16] C. Yin, Y. Zhu, J. Fei, and X. He, "A deep learning approach for intrusion detection using recurrent neural networks," *Ieee Access*, vol. 5, pp. 21 954–21 961, 2017.

[17] L. Nicholas, S. Y. Ooi, Y. H. Pang, S. O. Hwang, and S.-Y. Tan, "Study of long short-term memory in flow-based network intrusion detection system," *Journal of Intelligent & Fuzzy Systems*, pp. 1–11, 2018.

[18] A. F. M. Agarap, "A neural network architecture combining gated recurrent unit (gru) and support vector machine (svm) for intrusion detection in network traffic data," in *ACM 10th International Conference on Machine Learning and Computing*, 2018.

[19] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *International conference on machine learning*, 2013.

[20] O. Kuchaiev and B. Ginsburg, "Factorization tricks for lstm networks," *arXiv preprint arXiv:1703.10722*, 2017.

[21] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[22] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy, "Hierarchical attention networks for document classification," in *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2016, pp. 1480–1489.

[23] Y. Kim, C. Denton, L. Hoang, and A. M. Rush, "Structured attention networks," *arXiv preprint arXiv:1702.00887*, 2017.

[24] A. Brown, A. Tuor, B. Hutchinson, and N. Nichols, "Recurrent neural network attention mechanisms for interpretable system log anomaly detection," *arXiv preprint arXiv:1803.04967*, 2018.

[25] M. Zhu, K. Ye, Y. Wang, and C.-Z. Xu, "A deep learning approach for network anomaly detection based on amf-lstm," in *IFIP International Conference on Network and Parallel Computing*. Springer, 2018.

[26] Z.-Q. Qin, X.-K. Ma, and Y.-J. Wang, "Attentional payload anomaly detector for web applications," in *International Conference on Neural Information Processing*. Springer, 2018.

[27] P. Casas, J. Mazel, and P. Owezarski, "Unsupervised network intrusion detection systems: Detecting the unknown without knowledge," *Computer Communications*, vol. 35, no. 7, pp. 772–783, 2012.

[28] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[29] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, "Convolutional sequence to sequence learning," in *34th International Conference on Machine Learning*, 2017.

[30] J. Lei Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[31] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE CVPR*, 2016.

[32] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization." in *ICISSP*, 2018.

[33] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[34] J. Lafferty, A. McCallum, and F. C. Pereira, “Conditional random fields: Probabilistic models for segmenting and labeling sequence data,” 2001.

[35] X. He, R. S. Zemel, and M. Á. Carreira-Perpiñán, “Multiscale conditional random fields for image labeling,” in *IEEE CVPR*, 2004.

[36] A. McCallum and W. Li, “Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons,” in *the seventh conference on Natural language learning at HLT-NAACL*. Association for Computational Linguistics, 2003.

[37] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[38] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

[39] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, “Tensorflow: A system for large-scale machine learning,” in *OSDI 16*, 2016.

[40] R. Caruana, S. Lawrence, and C. L. Giles, “Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping,” in *Advances in neural information processing systems*, 2001.