Deep Sequence Models for Packet Stream Analysis and Early Decisions

Minji Kim* Dongeun Lee* Kookjin Lee† Doowon Kim‡ Sangman Lee§ Jinoh Kim*
Computer Science Department, Texas A&M University, Commerce, TX 75428, USA *
School of Computing and Augmented Intelligence, Arizona State University, Tempe, AZ 85281, USA †
Department of Electrical Engineering & Computer Science, University of Tennessee, Knoxville, TN 37996, USA ‡
Sysmate Inc., 1184 Beongil 41, Yuseong-gu, Daejeon 34109, Korea §

2022 IEEE 47th Conference on Local Computer Networks (LCN) | 978-1-6654-8001-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/LCN53696.2022.9843272

***Abstract*—The packet stream analysis is essential for the early identification of attack connections while in progress, enabling timely responses to protect system resources. However, there are several challenges for implementing effective analysis, including *out-of-order packet sequences* introduced due to network dynamics and *class imbalance* with a small fraction of attack connections available to characterize. To overcome these challenges, we present two deep sequence models: (i) a *bidirectional recurrent structure* designed for resilience to out-of-order packets, and (ii) a *pre-training-enabled sequence-to-sequence structure* designed for better dealing with unbalanced class distributions using self-supervised learning. We evaluate the presented models using a real network dataset created from month-long real traffic traces collected from backbone links with the associated intrusion log. The experimental results support the feasibility of the presented models with up to 94.8% in F1 score with the first five packets (k=5), outperforming baseline deep learning models.**

I. Introduction

Network-based attacks happen increasingly with growing volume and impacts. This becomes more critical with ever-growing connectivity with less secure Internet of Things (IoT) devices through wireless mobile communications. One of the essential functions to protect computing resources in the networking environment is the effective detection of network attacks. The approach to network attack detection can be either host-based or connection-based (i.e., host-independent), and the latter would be an attractive option with relatively low deployment and operation costs [1]. In addition to detection, the capability of timely responses is another vital requirement, so as to minimize detection deficit (the gap between the time to compromise and the time to defend) [2]. That is, the network should be able to prepare and enforce relevant countermeasures against individual attack activities before they are successfully executed. To this end, there would be at least two prerequisites: First, the class of attacks should be identified in the detection time to prepare relevant countermeasures (e.g., based on predefined security policies). Second, the detection process should be done as early as possible while the attack is active and in progress.

Recently a body of studies has employed machine learning (ML) techniques for detecting network attacks [3], [4]. Despite their promising detection performance reported, most (if not all) of previous studies assume the availability of complete connection records that are only available at the connection termination time. For instance, common public datasets (e.g., KDDCup 1999/NSL-KDD [5], UNSW-NB15 [6], and CIC-IDS2017 [7]) provide the complete connection information such as duration, aggregated bytes, and number of packets. Such reliance confines the capability of attack detection to postmortem analysis rather than live online detection for timely actions. Moreover, a large body of ML-based detection studies targeted anomaly detection (i.e., binary decision whether it is likely an attack or not); attack detection is a more complicated problem and requires the ability to identify the type of attacks (e.g., Web, DoS, and scanning attacks). Deep packet inspection is an effective method for identifying attack classes at the early stage of connections by applying pattern matching with predefined byte sequences ("signatures") [8]. However, any payload inspection approach is extremely expensive for analyzing packet payloads and limited to unencrypted packets only. Moreover, there is a trend of banning payload scanning with an increasing demand on privacy, which renders it unlawful to read even cleartext packets. In this study, we take an approach of *packet stream analysis* referencing statistical information of packet sequences, thus requiring *neither* packet inspection *nor* complete connection information.

For effective packet stream analysis, there exist several intricate challenges. First, the communication network is wild and network dynamics (e.g., routing oscillation) may lead to *incomplete* or *out-of-order* packet sequences in collection. For example, packet loss rates are non-negligible in reality, ranging from 0.01% to 20%, even in wired networking settings [9]. Packet reordering refers to the reception of packets in a different order than what were originally sent, and it is inevitable to see packet reordering in packet-switched networks due to several reasons, such as multi-path forwarding routes, parallelism, and packet loss and retransmission [10]. Possibly, recovering from packet reordering by referring to the sequence number field is an option, but it is a highly expensive operation with extra space and time requirements in an intermediate point in the network. Such dynamics is generally more critical in wireless mobile communications with greater loss and retransmission rates. Second, the measured data collection shows a high degree of *class imbalance* with only a small fraction of attack connections. We can also find significant variations even from the distribution of attack types (also observed from our dataset in Table II). Another challenge would be a trade-off between the number of packet stream variables to be managed and analyzed vs. the corresponding time/space complexity: for example, a greater set of variables in analysis may yield better performance, while imposing greater overheads and expenses.

Contributions: This paper sheds light on the design of deep learning models for effective packet stream analysis, which enables us to make timely responses against network attacks by identifying in their early stages. The main contributions are two-fold. First, we present two deep sequence models designed to overcome the aforementioned challenges: (i) a *bidirectional recurrent structure* designed for greater resilience to missing/out-of-order packet streams, and (ii) a *pre-training-enabled sequence-to-sequence structure* designed for creating consistent representations from unbalanced class distributions using self-supervised learning. Second, we share evaluation details with our observations and analysis. We conduct the evaluation using a real network dataset created from month-long real traffic traces collected from backbone links with the associated intrusion log. The experimental results show that our deep sequence models consistently outperform baseline deep learning models across diverse experimental settings, yielding up to 94.8% in F1 score with the first five packets (k=5).

The organization of this paper is as follows. In Section II, we provide the description of the problem and present the framework model designed to tackle the problem. We next introduce the dataset constructed from the Internet backbone traffic traces with the associated intrusion logs in Section III, and then present our deep sequence models developed to analyze the potentially imperfect and unbalanced packet stream data in Section IV. We share the experimental setting and results with our observations and implications in Section V, and provide a summary of closely related studies in Section VI. We finally conclude our presentation in Section VII, with the summary of the study and future directions.

II. Background

This section provides the description of the problem and our framework model designed to tackle the problem.

A. Problem Description

This study aims to accurately identify the type of the network connections in their early stages by analyzing the stream of packets. To formulate the problem, we employ the general concept of connections and flows defined for point-to-point communications. A connection consists of two flows for bidirectional communication (e.g., client-to-server and server-to-client directions). Let c_i be a connection and $\mathcal{C}$ the set of connections in a captured network trace. Then, the size of flow set $\mathcal{F}$ is twice the corresponding connection set size, i.e., $|\mathcal{F}| = 2 \times |\mathcal{C}|$. A common way of representing a flow (for early identification) is to associate it with a two-dimensional vector containing the *packet size* and *packet time* recorded for each packet within the flow, which has often been considered as time series analysis for traffic classification [11]. For flow f, let $b_j(f)$ be the number of bytes of the j-th packet and $t_j(f)$ be the inter-packet time between the (j-1)-th and j-th packets ($t_1(f)$ = 0). Let ψ be the function that transforms a flow into this representation: $\psi(f) = \langle (b_1(f), t_1(f)), ..., (b_n(f), t_n(f)) \rangle$, for flow f consisting of n packets.

For attack types, suppose m attack classes under consideration. A flow may either be a part of an attack connection or have nothing to do with any of attack classes (referred to as "background"). Let $\mathcal{U}$ be a set of different classes, $\mathcal{U} = \{A_i | 0 \leq i \leq m\}$, where A_0 is a class for background traffic while A_j ($j > 0$) is an attack class. In other words, a flow belongs to A_i ($0 \leq i \leq m$) depending on its class. The identification function $\phi(\cdot)$ takes the input representation of a flow $\psi(f)$ and determines its class, i.e., $\phi \circ \psi : \mathcal{F} \rightarrow \mathcal{U}$. We utilize standard metrics based on the confusion matrix, including accuracy and F1 score, to measure the performance of the identification function. For early identification, only the first k packets in a flow are monitored. The goal is then to maximize the identification performance while minimizing the number of packets referenced.

B. Overview of the Proposed Framework

We present our framework designed for providing effective packet stream analysis and early decisions for identifying attack classes. The framework defines a set of functional components to fulfill its mission, as illustrated in Figure 1.

- *Data Gen* creates normalized, labeled data instances by combining the given packet traces and the associated intrusion logs, which will be used for constructing the learning model. Attack flows are labeled with their specific attack class, while the rest of the flows are labeled as "Unlabeled";
- *Packet Stream Gen* groups a stream of packets based on the flow identifier information. The statistical packet stream information from the predetermined number of packets (k) on the same flow is measured and delivered to *Early Identifier*;
- *Pre-trainer* is equipped to perform self-supervised learning, which helps initialize the neural network in a way to stabilize the learning model from the impact by probabilistic skewness and class imbalance. Note that the label information is not referenced in the pre-training stage;
- *Main Trainer* learns the characteristics of individual attack classes by referencing the annotated class label information. The Pretrained model is loaded as the initialization of the Main Trainer, and the labeled samples are injected to construct the learning model for early attack identification;
- *Early Identifier* performs the identification of attack classes against the given data instance delivered from *Packet Stream Generator*. The learning model built by *Main Trainer* is loaded to this component for actual decisions. The early decision is then made by classifying each instance into one of the classes ($A_i \in \mathcal{U}$).

Basically, there are two parts in the presented framework. The data measurement part (*Data Gen* and *Packet Stream Gen*) will be described in Section III. In fact, the ML part is the core of the framework performing learning and decisions. As mentioned, there would be several challenges for realizing early decisions from potentially imperfect and unbalanced network packet data. We will present two deep sequence models developed for addressing those challenges in Section IV. The designed sequence-based models include a bidirectional recurrent structure built upon long short-term memory (LSTM) for resiliency against packet missing and reordering (designed

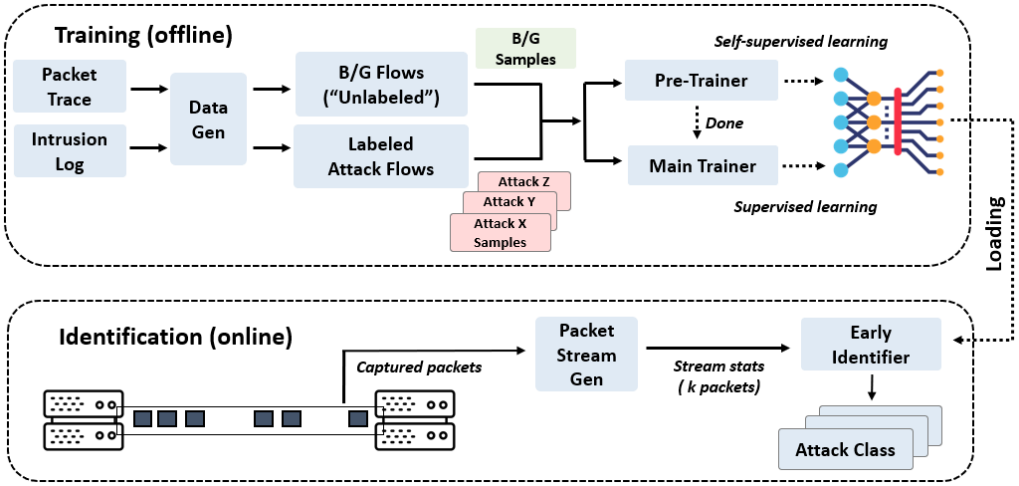

Fig. 1. Overview of the proposed framework for early attack identification, which analyzes packet streams using a time series-aware ML component for learning and classifying.

for *Main Trainer*), and a pre-training-enabled sequence-to-sequence structure based on the autoencoding architecture for stability from unbalanced class distributions (designed for *Pre-Trainer*).

III. Packet Stream Data Construction

To identify different classes of attacks, the availability of annotated datasets (with the label information) would be basically assumed for the unique representation of each class. While public datasets, such as KDD Cup 1999 [12], UNSW-NB15 dataset [6], and CIC-IDS2017 [7], extracted from traffic traces have been exhaustively utilized for many intrusion detection studies, those datasets basically contain the information of flows that can be only collected *after* the completion of communication. Hence, such datasets are rather useful for postmortem analysis than for early detection of ongoing malicious flows.

In this work, we construct a fine-grained packet-level dataset including the size (in bytes) and time information of individual packets (i.e., constructing $\psi(f)$) by combining public traffic traces with corresponding intrusion detection logs. Specifically, we utilize the real network traces and intrusion logs collected from backbone links in Japan [13], [14]. The traffic trace contains TCP/IP packet header information in a pcap file, while the associated intrusion log is provided in a comma-separated values (CSV) file with the attack information inferred by multiple detectors. Each pcap file is a recording of 15-minute traffic collected on a specific day. A previous study in [15] constructed network data by combining these traffic traces and intrusion logs, but the resulting dataset contains NetFlow-like statistical information made only available at the connection release time. In this work, we construct packet stream data to develop the function for the early identification of network attacks.

Here, we describe the details about our data construction process. Basically, the intrusion log contains the flow identification information ("5-tuple"), including the IP address pair, source/destination port numbers, and the transport-layer protocol (e.g., TCP). The log also provides the attack class information, including HTTP-relevant attacks ("HTTP"), heavy hitter traffic ("Alpha"), point-to-multipoint anomalies ("Multi"), denial-of-service attacks ("DoS"), port scanning ("PortScan"), network scanning ("ICMPScan", "TCPScan", and "UDPScan"), and tunneling behaviors ("IPv6Tunnel") [14]. As can be seen from Table I, a specific attack is categorized to one of the attack classes based on its intrinsic characteristics.

TABLE I
Attack Class Information Defined in the Intrusion Log

Class	Attack Prefixes
HTTP	"alphflHTTP", "ptmpHTTP", "mptpHTTP", "ptmplaHTTP", "mptplaHTTP"
Multi	"ptmp", "mptp", "mptmp"
Alpha	"alphfl", "malphfl", "salphfl", "point_to_point", "heavy_hitter"
IPv6_Tunnel	"ipv4gretun", "ipv46tun"
Port scan	"posca", "ptpposca"
ICMP_scan	"ntscIC", "dntscIC"
UDP_scan	"ntscUDP", "ptpposcaUDP"
TCP_scan	"ntscACK", "ntscSYN", "sntscSYN", "ntscTCP", "ntscnull", "ntscXmas", "ntscFIN", "dntscSYN"
DoS	"DoS", "distributed_dos", "ptpDoS", "sptpDoS", "DDoS", "rflat"

Using the flow identifier, the traffic trace is combined with the associated intrusion log. If the flow from the traffic trace has a match to a log entry, then the flow is marked as an attack with the class information ($A_i \in \mathcal{U}$ and $i \neq 0$) . If no match is found, then the combining procedure falls back to 4-tuple (excluding the source port number information from 5-tuple) and performs the same combining task. In case of no match again from the fall-back process, the flow is assumed to be benign, and thus, background traffic ($A_0 \in \mathcal{U}$). The statistical packet stream information is then recorded with the class information for each flow in the traffic trace. In this work, we consider TCP flows only since it is safe to construct a flow using connection management flags (e.g., a SYN packet for a

TABLE II
CONSTRUCTED DATASET WITH CLASS LABELS (09/01/2020-09/30/2020)

Class	k=3	k=5	k=10	k=20
HTTP	5,006	3,639	2,502	2,208
Multi	19,478	12,410	6,236	2,866
Alpha	136,070	131,888	27,527	5,525
Port_scan	1	1	0	0
DoS	42	33	3	3
Unlabeled	17,963,835	13,252,563	4,075,647	940,004

new flow start).

Table II shows the extracted flow information from the 25-day network traffic collected in September 2020, except five days due to unavailability (3rd, 14th, 27th, 28th, and 29th). In the table, we use the label of "Unlabeled" to indicate background traffic (i.e., flows that have not been assigned to any of the attack classes during the combining phase). From the one-month trace, we observe a set of flows in five attack classes in addition to background traffic, while there exists no flow for the rest of the attack classes. Note that it is natural that the number of flows decreases with greater k values, because there can be a subset of flows consisting of a small number of packets only, and any flow with less than k packets should be excluded from the collection.

Finally, the extracted data samples are pre-processed before being fed to ML algorithms, which largely includes rescaling and data normalization. In particular, we observed high skewness from both the byte and inter-packet time fields. To resolve this, in the rescaling process, a logarithm function was applied to take the skewness of distributions into account. Then, data normalization was performed with the min-max scaling function. The constructed dataset (in Table II) is publicly available online through https://github.com/dcstamuc/PacketStreamDataCollection.

IV. METHODS – DEEP SEQUENCE MODELS

The primary goal of this study is to develop an effective packet stream analysis tool that provides accurate early attack identification from potentially incomplete, skewed packet stream datasets with out-of-order sequences and unbalanced class distributions. In this section, we first describe the overview of deep learning for time series analysis, and then present our deep sequence models designed for addressing above challenges: (i) bidirectional long short-term memory (LSTM) as an implementation of bidirectional recurrent structure, and (ii) sequence-to-sequence autoencoder as an implementation of pre-training-enabled sequence-to-sequence structure.

A. Deep Learning for Time Series Analysis

Considering a sequence of packets in a flow as the input variable and an attack class as the target variable, we can formulate the identification problem as a classification problem in a supervised learning setting. In particular, we aim to learn a function ϕ_Θ, which predicts the attack class given the first k packets of a flow, $A_i = \phi_\Theta(\psi_k(f))$, where Θ is a set of model parameters to be learned; and $\psi_k(\cdot)$ maps to k tuples.

Among various forms of ϕ_Θ, we choose recurrent neural networks (RNNs) because they are specifically designed for processing time series data by utilizing shared and recurring units [16]. At time index t, RNNs operate on an input x_t and update hidden states $\boldsymbol{h}_t = (h_t^{(1)}, h_t^{(2)}, \ldots)$ by applying a recurring function,

$$\boldsymbol{h}_t = \mathrm{RNN}(x_t, \boldsymbol{h}_{t-1}; \Theta_{\mathrm{RNN}}),$$

where Θ_{RNN} denotes the model parameters of RNN and $h_t^{(\ell)}$ denotes a hidden state at the ℓth RNN layer. For the classification task, a classifier network takes (a part of) the last hidden state $\boldsymbol{h}_{\mathrm{last}}$ to make a prediction on a probability distribution over a set of classes, i.e., $\boldsymbol{p} = \mathrm{c}_\xi(\boldsymbol{h}_{\mathrm{last}})$, where $\boldsymbol{p} = [p_0, \ldots, p_m]^\mathsf{T}$ denotes a vector of probability masses, each p_i of which denotes the probability that the flow is classified as the ith label; and c_ξ is a classifier network with a set of trainable parameters ξ. For a given $\boldsymbol{p}$, the attack (or background) class A_i is determined by the index i that satisfies $p_i \geq p_j$ for all $j \in \{0, \ldots, m\}$.

a) Long Short-Term Memory: As standard RNNs typically suffer from vanishing or exploding gradient problems [17], modern RNN architectures such as LSTM [18] and gated recurrent unit (GRU) [19] were proposed as gating mechanisms to mitigate such problems by creating computational paths whose gradients do not vanish or explode for a long period. In this study, we choose LSTM over GRU as LSTM is known to be strictly more expressive, outperforming GRU in many complex applications [20]. LSTM resolves vanishing/exploding gradient problems by adding LSTM cells. In addition to the standard RNN hidden states $\boldsymbol{h}_t$, LSTM cells have a cell state $\boldsymbol{s}_t$, which is updated by an internal recurrence and creates paths where the gradient can flow over longer duration. Then the gating mechanisms control the flow of information by employing three gates: input, output, and forget gates. The input gate controls the extent to which new input data is transferred to update $\boldsymbol{s}_t$, the output gate controls the extent to which the LSTM cell output flows out to the output (the hidden state), and the forget gate controls the extent to which $\boldsymbol{s}_t$ remains unchanged. The gating mechanisms allow important information to last longer while forgetting less significant information.

b) Bidirectional LSTM: In our task of the attack classification, the performance of the task depends not only on understanding "causal" relationship of packets in the input sequence, but also on extracting other useful features that would otherwise be overlooked by only focusing on causality. However, a regular LSTM only employs a past history to extract features. To address this issue, bidirectional LSTM [21] has been studied and has demonstrated its effectiveness in many applications [22]. Thus, in this study, we consider the bidirectional LSTM, which combines two LSTM architectures: one moving forward from the beginning to the end of the input sequence and the other moving backward from the end to the beginning of the input sequence. With this architecture, we anticipate that the model can handle a flow better even if packets arrive out of order.

c) Training: As the classifier network is designed to produce a probability that the input flow is categorized into

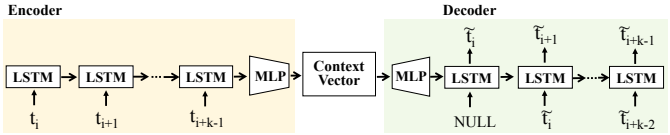

Fig. 2. Seq2Seq autoencoder for self-supervised learning.

each class, the last component of the classifier is a softmax layer, which turns its input vector $\boldsymbol{o}$ into the probability vector $\boldsymbol{p}$ such that $p_i = \exp(o_i)/\sum_i \exp(o_i)$. Then, the model can be trained by minimizing the cross-entropy loss:

$$L_{\mathrm{CE}} = -\frac{1}{n_{\mathrm{train}}} \sum_{j=1}^{n_{\mathrm{train}}} \sum_{i=0}^{m} y_i \log(p_i),$$

where $y_i \in \{0, 1\}$ is an indicator denoting if the index i is the correct classification, and n_{train} is the number of training samples.

B. Self-supervised Learning

In many applications, the distribution of classes of interest could be biased or skewed, and training a classifier directly could fail to capture an underlying distribution correctly. To mitigate this class imbalance issue, a training algorithm that does not require the knowledge of class labels can be employed to pre-train the classifier. As pointed out in the state-of-the-art unsupervised learning study (contrastive-loss-based models, e.g., [23]), a main purpose of unsupervised learning is to pre-train a model, which is in turn fine-tuned with supervised learning. We follow this approach in this study.

a) Sequence-to-sequence autoencoder: Autoencoder is a feed-forward neural network, which attempts to copy its input sequence to its output sequence. We consider a type of autoencoder, called sequence-to-sequence (Seq2Seq) autoencoder model, whose input and output are sequences. In general, the Seq2Seq model consists of three parts: an encoder, a decoder, and a context vector. To handle the sequence, the encoder and the decoder are typically modeled as RNN-type neural networks. The encoder processes an input sequence and produces the context vector, which would contain the most salient features of the input sequence. Figure 2 depicts the network architecture of the Seq2Seq autoencoder model. Internally, the multilayer perceptron (MLP) component inside the encoder compresses the output of the LSTM to produce the context vector projected in a lower-dimensional space, and the other MLP at the decoder performs the decompression.

We now formally describe the Seq2Seq model. Specifically, the model can be written in a form:

$$\tilde{\psi}_k(f) = g_{\mathrm{dec}}(g_{\mathrm{enc}}(\psi_k(f); \Theta_{\mathrm{enc}}); \Theta_{\mathrm{dec}}),$$

where g_{enc} and g_{dec} denote the encoder and the decoder, respectively; Θ_{enc} and Θ_{dec} denote the model parameters for the encoder and the decoder, respectively; $\tilde{\psi}_k(\cdot)$ denotes the reconstruction of $\psi_k(\cdot)$. The encoder takes the input sequence and produces the context vector c:

$$c = g_{\mathrm{enc}}(\psi_k(f); \Theta_{\mathrm{enc}}).$$

The decoder performs the reverse action of the encoder,

$$\tilde{\psi}_k(f) = g_{\mathrm{dec}}(c; \Theta_{\mathrm{dec}}),$$

where the output approximates the input $\tilde{\psi}_k(f) \approx \psi_k(f)$.

b) Training: The goal of the training is to find sets of model parameters $(\Theta_{\mathrm{enc}}, \Theta_{\mathrm{dec}})$ that minimize MSE between the output sequence and the input sequence:

$$L_{\mathrm{MSE}} = \frac{1}{n_{\mathrm{train}}} \sum_{i=1}^{n_{\mathrm{train}}} \left\| \psi_k(f) - \tilde{\psi}_k(f) \right\|_2^2.$$

Note that this approach can be categorized as self-supervised learning, as it does not require additional labels for training, except the input variable. Once the training of the Seq2Seq autoencoder is finished, we take the network weights and biases of the encoder as initial values of network weights and biases of the LSTM network for the classification, i.e., $\Theta_{\mathrm{RNN}} = \Theta_{\mathrm{enc}}$. After that, we continue training the LSTM network to minimize the cross-entropy explained in Section IV-A.

V. Evaluation

In this section, we share the evaluation results with our observations and implications. We first describe the experimental setting for compared models, data preparation, and evaluation metrics. Then we report the experimental results, which demonstrate the feasibility of our deep sequence models for early decisions with improved performance compared to the baseline models.

A. Experimental Setting

We evaluate a set of deep learning models for packet stream analysis in the context of network attack identification. As the baseline, we consider two neural network models: (i) an MLP model that takes a vector containing the packet size and time without the notion of time sequences ("MLP") and (ii) a unidirectional LSTM model widely applied for time series analysis ("FWD"). As the realization of the proposed models, we implement two deep sequence models: (iii) the bidirectional LSTM model ("BI") and (iv) the unidirectional LSTM model pre-trained with Seq2Seq autoencoder ("SEQ"). Note that we configure FWD with the identical neural network architecture to SEQ, with the same level of optimizations for fair comparison; hence, SEQ can be regarded as a pre-trained version of FWD with self-supervised learning. For BI, we test models with an additional fully-connected layer that produces a hidden vector, which has the same size as the context vector. Internally, we use `ReLU` for MLP and `tanh` for LSTM for non-linear activation. For each ML model, we utilize checkpointing to search a model instance with the greatest validation accuracy over 1,000 epochs with the learning rate 10^{-3}.

As shown in Table II, the created dataset mainly contains three attack classes (HTTP, Multi, and Alpha). We composed the ML models to classify the traffic into four classes (i.e., three attack classes and the background). For training, $\min(10000, x)$ number of instances were randomly chosen from each class pool, where x is the actual size of the pool. Hence, the sampled dataset is intrinsically unbalanced with less than or equal to 10,000 samples, particularly if k gets larger. The validation set contains 200 instances for each

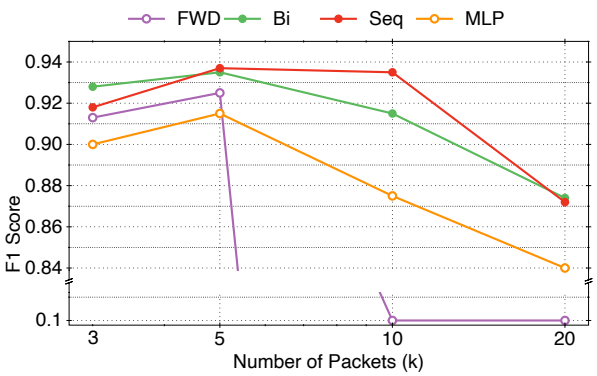

Fig. 3. Performance comparison: BI and SEQ consistently outperform MLP and FWD across different k values, yielding up to 93.5% (BI) and 93.7% (SEQ) with k=5. Note that FWD collapses to around 10% when $k \geq 10$ (thus showing the broken line).

TABLE III
BEST-PERFORMING CONFIGURATION FOR EACH MODEL ($k = 5$)

Model	HLAYER	CV	HDIM	F1 Score	FPR
MLP	3	–	40	0.937	0.13
FWD	3	–	40	0.945	0.07
BI	3	10	40	0.947	**0.05**
SEQ	3	10	20	**0.948**	0.08

class, and the testing set includes 100 instances per class. To ensure no-overlap among training, validation, and test sets, the validation and test samples had been drawn before composing the training set.

To measure model performance, we basically refer to the confusion matrix consisting of TP (True Positive), FP (False Positive), FN (False Negative), and TN (True Negative). We measure the identification performance using *F1 score*, since the metric of *accuracy* may lead to a biased outcome under a setting with unbalanced classes (same as our setting). The metric of F1 score is defined as: F1 score $= \frac{2TP}{2TP+FP+FN}$, and hence, the model performs better if the score is higher. Since the attack identification is *not* a binary classification problem, we calculate F1 score based on the number of true instances for each class (i.e., *Macro Avg.* F1 score). We additionally report false positive rate (FPR) defined as: FPR $= \frac{FP}{FP+TN}$. In our setting, a false positive event refers to a case where Unlabeled (A_0) is classified into one of the attack classes ($\{A_i | i \neq 0\}$).

B. Experimental Results

We first perform a fair comparison under the assumption of the identical configuration setting for the number of hidden layers (HLAYER) and the hidden dimension size (HDIM): HLAYER=3 and HDIM=20. Additionally, we set the size of context vector (or hidden vector) (CV) to three for BI and SEQ as their default values (i.e., CV=3). We then report the configuration of each model yielding the best F1 score with the corresponding FPR. Finally, we analyze the performance of BI and SEQ with different values for CV and HDIM.

TABLE IV
PERFORMANCE (F1 SCORE) UNDER DIFFERENT CONFIGURATIONS
(**bold**=BEST, *italic*=BEST FOR THE OTHER MODEL)

Model	k=3	k=5	k=10	k=20
BI(3,20)	0.928	0.935	0.915	0.874
BI(5,20)	0.935	0.935	0.908	0.878
BI(10,20)	0.938	0.945	0.935	0.888
BI(3,40)	**0.948**	0.943	0.930	**0.910**
BI(5,40)	0.923	0.935	**0.938**	0.896
BI(10,40)	0.933	*0.947*	0.923	0.883
SEQ(3,20)	0.918	0.937	0.935	0.872
SEQ(5,20)	0.915	0.947	0.918	0.869
SEQ(10,20)	0.915	**0.948**	0.928	*0.901*
SEQ(3,40)	*0.925*	0.943	**0.938**	0.893
SEQ(5,40)	0.915	0.925	0.906	0.893
SEQ(10,40)	0.915	0.922	0.916	0.866

Figure 3 shows the classification performance in F1 score under the fair comparison setting. From the figure, we can see that our deep sequence models (BI and SEQ) substantially outperform MLP consistently across different k's. Interestingly, the unidirectional LSTM model (FWD) collapses to 0.1 when k=10, implying high sensitivity and instability of the model when the sampled dataset gets more unbalanced. BI and SEQ are capable of providing early identification of attack flows with fairly acceptable performance even with three packets (k=3), while monitoring the first five packets (k=5) escalates the identification performance to 93.5% (BI) and 93.7% (SEQ). One interesting observation is that increasing the number of packets ($k \geq 10$) does not help improve the identification performance, which will be discussed in Section V-C.

We next compare the best performance of the models. In fact, we have performed extensive hyper-parameter-search on a grid and here we report the results of the best configurations with respect to F1 score when k=5 (as observed in Figure 3). Table III provides the performance with F1 score and FPR for each model. In the table, CV is not applicable for MLP or FWD. The results show that SEQ is slightly better than the others, while BI shows the lowest FPR. Although the performance gap between the baseline models (MLP and FWD) and our deep sequence models (BI and SEQ) is small here, BI and SEQ perform more consistently, which is a crucial factor in actual deployment.

Lastly, we provide further results for BI and SEQ under different CV and HDIM settings. For the exposition purpose, we name a model with a suffix of (CV, HDIM): for example, BI(3,20) refers to the BI model with CV=3 and HDIM=20. The results in Table IV suggest a room to optimize the models with non-negligible improvements compared to the default setting, i.e., (CV=3, HDIM=20). In the table, the bold-faced result indicates the best performance among the entire settings for a given k. We also italicize the best performance of the other model for the comparison of the two models (i.e., BI and SEQ). For example, BI(3,40) yields the best performance (and bold-faced) when $k = 3$, while SEQ(3,40) works the best among the SEQ settings (and italicized). Overall, BI(3,40) performs consistently, yielding over 91% without any signif-

TABLE V
F1 SCORE WITH DIFFERENT DATASET CONFIGURATIONS (k=5): 1,000 SAMPLES FOR SET1 & SET3 AND 10,000 FOR SET2 & SET4

	Week (09/01 – 09/07)		Month (09/01 – 09/30)	
Model	Set1	Set2	Set3	Set4
BI(10,20)	0.913	0.968	0.947	0.945
SEQ(10,20)	0.895	0.955	0.948	0.948

icant variations across different configurations. SEQ works well by and large but shows some degraded performance with the longest sequence (k=20), presumably due to the increased complexity of the reconstruction.

C. Discussion

In Figure 3, we observed that increasing the number of packets ($k \geq 10$) does not improve any identification performance, which is somewhat counter-intuitive since monitoring more packets could give further information about the communication. Regarding this, we can get some clue from previous studies. A pioneering study in [24] claims that the first four packets of a TCP connection would be sufficient to characterize network applications, and the authors observed that monitoring more than four packets even results in the degradation of identification performance. A recent study in [25] also concludes that keeping track of more packets does not help the prediction accuracy for traffic classification, in which k ranged between 30 and 60.

Another interesting observation in our evaluation is the impact of training samples on identification performance. As described, we randomly sampled $\min(10000, x)$ number of instances from each class pool for the evaluation (including 200 for validation and 100 for testing), where x is the actual size of the pool. In Table V, we organize four different sets for evaluating: Set1 has 1,000 instances sampled and Set2 10,000 instances sampled from the first week, while Set3 has 1,000 samples and Set4 10,000 samples from the whole month. With the week-long datasets (Set1 and Set2), we can clearly see the benefit of the increased training set size. In contrast, the month-long setting does not give any progress even with the larger dataset. One possible explanation is that using 10,000 instances would not be enough to capture the distribution of data from the month-long pool, while that same number would suffice for the relatively small week-long pool. This would also be a reason why Set2 shows the better performance than Set4. However, if we have to draw a less number of instances (i.e., 1,000 samples for Set1 and Set3), the result shows that sampling from the whole month performs better; presumably, it would be beneficial to choose samples from the larger (month-long) pool if we have to choose only a small number of instances. We observed almost the same patterns from different BI and SEQ configurations. In fact, this study limits the number of samples to 10,000 to keep computational complexity manageable but evaluating the models with an extensive set of data samples from a longer-term capture would be interesting to better understand the impact of data quantity and quality.

In this study, we consider the number of packets as the criterion for the early attack identification. We note that this criterion is different from the wall-clock time and therefore intervals between packets are not considered in the training. It would be interesting to see if the integration of intervals in the training would increase the identification performance.

VI. RELATED WORK

Intrusion detection is a typical second line of defense mechanism, which can broadly be classified into two groups of signature-based and anomaly-based approaches [8]. Conventional intrusion detection tools, such as Snort (https://www.snort.org/) and Zeek (https://zeek.org/), have the capability of detecting unwanted connections in real time often using textual byte patterns. Given the wider reliance on payload encryption, however, those signature-based approaches would be less attractive [8]. Moreover, there is an increase of privacy disputes and legal regulations, and the payload inspection may not be a valid option, particularly at a certain network point.

A body of studies investigated various ML approaches for detecting anomalies from network traffic traces [3], [4]. Despite substantial improvement with brand-new methods, anomaly detection is inherently limited to the binary decision without providing information specific to attack classes. Additionally, existing studies largely rely on complete connection information, which only becomes available after the communication is over. Several studies [26], [27], [7] focused on attack identification (rather than anomaly detection); however, they still rely on the full picture of connections without the early identification feature.

Early detection has been a keyword for other application domains as well such as traffic classification [25], [28], fake news detection [29], and fraud detection [30]. Traffic classification is also a crucial function in network monitoring and management for traffic engineering, resource provisioning, Quality-of-Service (QoS) preservation, and security purposes. Several studies tackled the problem of traffic classification based on packet stream analysis, and hence, their proposed method might provide the function of early classification of network flows by referencing the first few packets. However, traffic classification would be less impacted by out-of-order packet sequences: for example, there is a clear distinction on packet spacing patterns between loss-sensitive applications (e.g., file transfer) vs. time-bounded applications (e.g., virtual conferencing). Several studies, including [29], [30], [31], [32], [33], [34], focused on fake news/rumor detection, with the goal of minimizing the impact of adversarial events by detecting them in the premature stage. Unlike our problem, however, they do not need to consider the possibility of inconsistency in data sequences owing to the assumption of posted messages on social media with time stamps.

VII. CONCLUSION

This paper presented the design and evaluation of deep sequence models designed for effective packet stream analysis to achieve accurate identification of network attacks in a timely manner. With the rationale of network dynamics (causing out-of-order packet sequences) and class imbalance (resulting in learning inconsistent representations), we designed two deep sequence models based on a bidirectional structure (for the

former concern) and a pre-training-enabled Seq2Seq structure (for the latter concern). To evaluate the presented models, we construct a packet-level dataset from month-long real Internet backbone traces with the associated intrusion log collection. Our experimental results support the feasibility of the presented deep sequence models, showing up to 94.8% (F1 score) for identifying three different attack types (with one background traffic class). The results also suggest that monitoring the first five packets (k=5) would suffice, enabling timely responses against network threats while the connection is still active and in progress.

There are several directions for further investigations. We examined the two deep sequence models independently, but ultimately the models would be combined to expect a synergistic benefit, which will be the next step of this study. We also plan to evaluate the presented models with other packet traces publicly accessible with the associated ground truth attack information (e.g., UNSW-NB15 collected on a simulated setting) for extensive analysis and optimizations.

ACKNOWLEDGMENT

The authors would like to express our gratitude to the anonymous reviewers for their constructive feedback. This work was supported in part by the Texas A&M University Presidential GAR Initiative program.

REFERENCES

[1] Q. Zou, A. Singhal, X. Sun, and P. Liu, "Deep learning for detecting network attacks: An end-to-end approach," in *IFIP Annual Conference on Data and Applications Security and Privacy (2021)*, pp. 221–234, Springer, 2021.

[2] A. Oprea, Z. Li, R. Norris, and K. Bowers, "Made: Security analytics for enterprise threat detection," in *Proceedings of the 34th Annual Computer Security Applications Conference (2018)*, pp. 124–136, 2018.

[3] N. Moustafa, J. Hu, and J. Slay, "A holistic review of network anomaly detection systems: A comprehensive survey," *Journal of Network and Computer Applications*, vol. 128, pp. 33–55, 2019.

[4] G. Fernandes, J. J. Rodrigues, L. F. Carvalho, J. F. Al-Muhtadi, and M. L. Proença, "A comprehensive survey on network anomaly detection," *Telecommunication Systems*, vol. 70, no. 3, pp. 447–489, 2019.

[5] L. Dhanabal and S. Shantharajah, "A study on nsl-kdd dataset for intrusion detection system based on classification algorithms," *Int. journal of advanced research in computer and communication engineering*, vol. 4, no. 6, pp. 446–452, 2015.

[6] N. Moustafa and J. Slay, "Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set)," in *Military Communications and Information Systems Conf. (MilCIS), 2015*, pp. 1–6, IEEE, 2015.

[7] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization.," in *ICISSP 2018*, pp. 108–116, 2018.

[8] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: techniques, datasets and challenges," *Cybersecurity*, vol. 2, no. 1, pp. 1–22, 2019.

[9] H. X. Nguyen and M. Roughan, "Rigorous statistical analysis of internet loss measurements," *IEEE/ACM Trans. on Networking*, vol. 21, no. 3, pp. 734–745, 2012.

[10] A. El-Atawy, Q. Duan, and E. Al-Shaer, "A novel class of robust covert channels using out-of-order packets," *IEEE Transactions on Dependable and Secure Computing*, vol. 14, no. 2, pp. 116–129, 2015.

[11] F. Pacheco, E. Exposito, M. Gineste, C. Baudoin, and J. Aguilar, "Towards the deployment of machine learning solutions in network traffic classification: A systematic survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1988–2014, 2018.

[12] M. Ahmed, A. N. Mahmood, and J. Hu, "A survey of network anomaly detection techniques," *Journal of Net. and Comp. Applications*, vol. 60, pp. 19–31, 2016.

[13] R. Fontugne, P. Borgnat, P. Abry, and K. Fukuda, "Mawilab: Combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking," in *Co-NEXT'10*, 2010.

[14] J. Mazel, R. Fontugne, and K. Fukuda, "A taxonomy of anomalies in backbone network traffic," in *2014 international wireless communications and mobile computing conference (IWCMC 2014)*, pp. 30–36, IEEE, 2014.

[15] J. Kim, C. Sim, and J. Choi, "Generating labeled flow data from mawilab traces for network intrusion detection," in *ACM Workshop on Systems and Network Telemetry and Analytics (2019)*, pp. 45–48, 2019.

[16] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," tech. rep., California Univ San Diego La Jolla Inst for Cognitive Science (1985), 1985.

[17] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

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

[19] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv:1406.1078*, 2014.

[20] D. Britz, A. Goldie, M.-T. Luong, and Q. Le, "Massive exploration of neural machine translation architectures," *arXiv:1703.03906*, 2017.

[21] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[22] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *IEEE int. conf. on acoustics, speech and signal processing (2013)*, pp. 6645–6649, 2013.

[23] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, "Momentum contrast for unsupervised visual representation learning," in *IEEE/CVF Conf. on Computer Vision and Pattern Recognition (2020)*, pp. 9729–9738, 2020.

[24] L. Bernaille, R. Teixeira, and K. Salamatian, "Early application identification," in *ACM CoNEXT conference*, pp. 1–12, 2006.

[25] S. Rezaei and X. Liu, "Multitask learning for network traffic classification," in *IEEE Int. Conf. on Computer Comm. and Net. (ICCCN 2020)*, pp. 1–9, 2020.

[26] C. Zhang, X. Costa-Pérez, and P. Patras, "Tiki-taka: Attacking and defending deep learning-based intrusion detection systems," in *ACM SIGSAC Conf. on Cloud Computing Security Workshop (2020)*, pp. 27–39, 2020.

[27] R. Vinayakumar, M. Alazab, K. Soman, P. Poornachandran, A. Al-Nemrat, and S. Venkatraman, "Deep learning approach for intelligent intrusion detection system," *IEEE Access*, vol. 7, pp. 41525–41550, 2019.

[28] Z. Chen, K. He, J. Li, and Y. Geng, "Seq2img: A sequence-to-image based approach towards ip traffic classification using convolutional neural networks," in *IEEE conf. on big data (big data 2017)*, pp. 1271–1276, 2017.

[29] Y. Liu and Y.-F. B. Wu, "Early detection of fake news on social media through propagation path classification with recurrent and convolutional networks," in *32th AAAI conference on artificial intelligence (2018)*, 2018.

[30] P. Zheng, S. Yuan, and X. Wu, "Safe: A neural survival analysis model for fraud early detection," in *AAAI Conf. on Artificial Intelligence (2019)*, vol. 33, pp. 1278–1285, 2019.

[31] X. Yang, Y. Lyu, T. Tian, Y. Liu, Y. Liu, and X. Zhang, "Rumor detection on social media with graph structured adversarial learning," in *International Joint Conf. on Artificial Intelligence, IJCAI 2020*, pp. 1417–1423, 7 2020.

[32] T. Bian, X. Xiao, T. Xu, P. Zhao, W. Huang, Y. Rong, and J. Huang, "Rumor detection on social media with bi-directional graph convolutional networks," in *Proceedings of the AAAI Conference on Artificial Intelligence (2020)*, vol. 34, pp. 549–556, 2020.

[33] P. Zheng, S. Yuan, X. Wu, J. Li, and A. Lu, "One-class adversarial nets for fraud detection," *AAAI Conf. on Artificial Intelligence*, vol. 33, pp. 1286–1293, Jul. 2019.

[34] B. Liu, Y. Li, Z. Sun, S. Ghosh, and K. Ng, "Early prediction of diabetes complications from electronic health records: A multi-task survival analysis approach," in *32th AAAI Conf. on Artificial Intelligence (2018)*, 2018.