Channel-Centric Spatio-Temporal Graph Networks for Network-based Intrusion Detection

Eduardo Santos Escriche, Jakob Nyberg, Yeongwoo Kim, and György Dán
Division of Network and Systems Engineering, KTH Royal Institute of Technology
Stockholm, Sweden
Email: {eduse, jaknyb, yeongwoo, gyuri}@kth.se

***Abstract*—The increasing frequency and complexity of cyberattacks against critical digital infrastructures require novel methods that can detect intrusions in a timely manner. Recent work has explored the use of Graph Neural Networks (GNNs) for network-based intrusion detection, adapting network traffic flow data to traditional GNN representations. In this work, we propose an alternative approach based on a novel combination of a graph representation of network traffic that represents communication channels as nodes and of a continuous temporal representation. Our proposed architecture, called Channel-Centric Spatio-Temporal Graph Networks (CCSTGN), can be trained using different learning strategies and is introduced together with a detailed data preprocessing strategy. We present an experimental evaluation of our proposed CCSTGN architecture using different learning strategies, from which we conclude that our proposal is able to outperform multiple existing GNN-based methods in terms of various classification metrics and that the data preprocessing procedure can be of significant importance for the performance of the models.**

***Index Terms*—graph neural networks, network intrusion detection, temporal graph networks, cybersecurity, machine learning**

I. Introduction

The frequency and sophistication of cyber-attacks have shown a rapid increase in the past decade [1]. Today, cyberattacks make use of a rich set of tactics, techniques, and procedures (TTPs), and typically involve various forms of illicit access to resources, *e.g.,* computers, over the network [2]. Illicit access may involve password cracking [3] or the use of stolen credentials obtained through social engineering [4]. Access to network resources, both illicit and legitimate, generates network traffic, resulting either in new traffic flows between hosts or in changes to the characteristics of existing flows.

Network-based Intrusion Detection Systems (NIDS) monitor network traffic as a way to identify illicit activity in a networked system. The two main categories of NIDS are signature based [5], [6] and anomaly based [7], [8]. Signature-based NIDSs produce alerts based on predefined rules or patterns, such as network addresses or packet contents. They have the benefit of being easy to operate and efficient in detecting known attacks, but they typically fail to detect unknown attack patterns for which signatures are not readily available. Anomaly-based NIDSs use a statistical characterization of baseline network activity for detecting deviations from the baseline. They would then generate alerts depending on the magnitude of the deviation. Anomaly-based NIDSs can detect previously unseen illicit activity for which there are no signatures, but they are instead prone to producing many false alerts, and thus require a significant amount of security expertise for maintaining real-time cyber situational awareness [9] and for generating actionable alerts in a timely manner [10].

Recent approaches to anomaly-based NIDSs incorporate deep learning as an attempt to reduce the number of false alerts while maintaining a high precision. Methods based on supervised learning include training deep neural networks [11] and Graph Neural Networks (GNNs) to classify flow level features [12], but their performance has been shown to be highly dependent on the dataset used for training and evaluation [13]. GNNs enable the inclusion of topological information about the network, which seems to allow better classification performance, but it is so far unexplored how to best represent the topological information for processing. Previous work has also noted that the temporal evolution of traffic flows can be equally important [13] as the topological information, but it is so far unclear how to best capture the temporal evolution of traffic flows.

In this paper, we propose a GNN-based NIDS that considers both the network topology and temporal evolution of traffic flows in its definition, seeking to overcome potential limitations of previous approaches. We define a set of methods to address those potential limitations, and make the following main contributions:

- We propose a novel combination of graph and temporal representations in the context of NIDS, and use it to develop a novel GNN architecture called Channel-Centric Spatio-Temporal Graph Network (CCSTGN) that is able to perform continuous-time dynamic graph learning.
- We provide a detailed description of our data preprocessing strategy, going beyond the explanations of most existing approaches and seeking to address some of their potential limitations.
- We carry out experiments with both supervised and unsupervised learning and apply a comprehensive set of evaluation metrics to analyze the suitability of our models for the task of intrusion detection[1].

The rest of this paper is organized as follows. Section II presents the related works that we consider in our analysis.

[1]Code is available at https://github.com/sanesedu/CCSTGN

Our problem formulation and analysis of previous graph and temporal representations, together with our proposed representations, are introduced in Section III. Based on those proposed representations, we describe the novel CCSTGN architecture in Section IV. Section V details various considerations related to our evaluation methodology. The numerical results obtained from the experiments that we conducted are presented in Section VI, together with our subsequent analysis. We conclude the paper in Section VII.

II. Related work

Multiple approaches to NIDS based on the application of GNNs have been proposed. 3D-IDS, introduced in [13], applies Graph Convolutional Networks (GCNs) for NIDS. The proposed approach aims at mitigating the entangled distribution of features that traffic flow data tends to present. The authors introduce a double feature disentanglement scheme, as well as a multi-layer graph diffusion method inspired by GIND [14] to better capture the considered spatio-temporal information.

EULER [15] is an approach for scalable dynamic link prediction and anomalous edge detection that explicitly considers temporal aspects of the data. The proposed architecture allows for the parallelization of the model through a leader and worker scheme. Workers compute node embeddings of a series of snapshots of the network activity using a GCN, and the leader then transforms the node embeddings into temporal embeddings that can be used for link prediction. A network's topology is thus encoded by a GCN at discrete time instants, and the temporal dynamics of changes in network connections are encoded by an RNN, such as an LSTM [16] network or GRU [17] network. EULER then aims to learn a probability function over the state conditioned on previous states of the considered temporal graph, in order to determine the likelihood of a link occurring at a later point in time.

E-GraphSAGE [12] adapts the GraphSAGE [18] approach to leverage both the edge features of the graph and its topological information with the goal of improving on the performance of existing methods for NIDS in the context of Internet of Things (IoT) networks. It uses an inductive learning approach, where the node embeddings are initialized to a constant vector, the message-passing process is modified to operate over edge features, and the final edge embeddings are computed by concatenating the node embeddings of the incident nodes.

The works presented above consider a host-centric graph representation and either ignore or apply discrete-time representations for the temporal aspects of the network traffic data. Our proposal instead combines an alternative channel-centric graph representation and a continuous-time temporal representation with the goal of addressing the potential limitations of the considered approaches.

In regards to the graph representation, our approach is similar to the line graph representation considered in [19]. Our proposed temporal representation is inspired by Temporal Graph Networks (TGN) [20], which provide a continuous-time representation for temporal graphs. TGN follows an encoder-decoder approach, in which the encoder maps the dynamic graph to node embeddings, and the decoder receives as input one or more node embeddings, which it uses in order to perform a task-specific prediction. TGN is based on a memory module, whose state contains a vector representation for each node that the model has previously observed. The memory state of a node is updated for each event involving that node, based on a series of modules that compute and aggregate the corresponding messages to the node. Lastly, TGN also includes an embedding module that is used to generate temporal embeddings of the nodes at a particular time, while attempting to avoid the problem of *memory staleness*, which occurs when the absence of events at a node for an extended period of time results in the node's memory becoming outdated.

III. Problem Formulation and Preliminaries

A. Detection Problem

We consider a networked system, where we observe a sequence of network flows. The i-th flow, $i \in \mathbb{Z}_{\geq 0}$, is characterized by three components: its flow ID v_i (the IP addresses of the involved devices), its timestamp $t_i \in \mathbb{R}$, and an N dimensional feature vector $\mathbf{f}_i \in \mathbb{R}^N$. Depending on the dataset, the feature vector may contain data such as the number of packets in the flow, the number of bytes or the duration of the communication.

Given the flows $1, \dots, i-1$, we define the intrusion detection problem as performing binary classification for traffic flow i, indicating whether it is benign or part of an intrusion attempt. The predicted output for flow i is computed based on information from the previously observed flows and corresponds to a probability distribution over the defined classes.

The architecture proposed in this paper is designed to support different learning strategies, and we include variants both trained with supervised and self-supervised learning in the evaluation. Our proposed architecture is also applicable to both binary and multi-class classification, but our experimental evaluation is limited to the binary case.

B. Channel-centric graph representation

Most GNN-based approaches for NIDS, *e.g.*, [12], [13], [15], represent the network traffic as a graph by assigning hosts in the network to nodes in the graph and communication between hosts to edges between the corresponding nodes. Mapping hosts to nodes is common in the context of graph-based network traffic analysis, such as for multipath TCP [21] or wireless network optimization [22]. However, we suspect it might not be the optimal network representation for network intrusion detection. When defining a NIDS based only on network information and traffic data, there tends to be little information about the hosts in the network besides their IP addresses.

We conjecture that defining the nodes of the graph representation using entities we lack data for, while the edges encode all the available information about the system, could be detrimental to the effectiveness of the representation in the

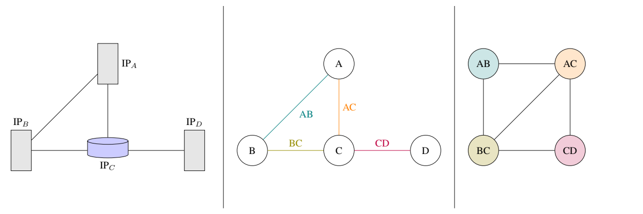

Fig. 1. Illustration of two methods for representing a computer network as a graph. (Left) Network structure. (Middle) Host-centric representation, where hosts are mapped to nodes. (Right) Channel-centric representation, a line graph of the network.

context of network intrusion detection. The lack of information about the state of the nodes in the graph also impacts the adaptation of GNN models for the task of NIDS, since the initialization and aggregation of information in GNNs usually happens over the nodes. As a result, when adapting or defining GNNs for NIDS with host-centric representations, some contrived design choices tend to appear, *e.g.*, calculating edge predictions based on node representations that were generated from edge information, as done in E-GraphSAGE [12]. We reason that the calculation can be simplified by shifting focus away from the hosts entirely.

The alternative approach we put forward, which we name *channel-centric*, is to consider the communication channels between devices as the main entities of the learning problem and thus define them as the nodes of the graph representation, with two nodes being adjacent if they have a host in common. This approach corresponds to defining the *line graph* of the host-centric representation graph. Fig. 1 shows the *host-centric* and *channel-centric representations* of a network with four nodes.

With this alternative representation, the traffic flows in the input data correspond to observations of states of nodes in our graph at different times. Information is concentrated in the nodes of the graph instead of the edges, which encode the spatial structure of the network. Therefore, we conjecture that this representation is more appropriate for building GNN-based NIDS since existing implementations tend to prioritize the calculation and analysis of node embeddings, thus simplifying their usage in the domain of NIDS. On the downside, this representation can become more computationally expensive than the host-centric representation when the number of communication channels is significantly larger than the number of devices in the network.

C. Temporal representation

Several approaches presented in Section II, such as E-GraphSAGE [12], do not explicitly consider the temporal aspects of the network in their models. Instead, they define a single static graph for the entire timespan the dataset covers where the host devices correspond to the nodes of the graph, and all the flows are converted into edges between their corresponding host devices. This representation ignores the temporal information associated with the flows, essentially allowing the model to look into the future. In addition, combining the entire timespan into a single graph can potentially result in a multigraph with a high average degree among the nodes. Large node neighbourhoods can negatively impact the performance and accuracy of GNN models since the node representations need to encode a a lot of information [18].

Some NIDS approaches, *e.g.*, EULER [15] or 3D-IDS [13], consider the temporal information associated with the data. These approaches model the temporal evolution of the flows by using discrete-time dynamic graphs, *i.e.*, they define a dynamic graph as an equally spaced series of timestamped static graphs or "snapshots". Using equally spaced snapshots might not be well-suited for the definition of a NIDS since traffic flows are not necessarily uniformly spaced in time, which could prove to be a limiting factor for the performance of the models, *e.g.*, when a flow is grouped in a snapshot that does not contain its time-wise closest contextual information.

Our proposed approach is a continuous-time dynamic graph formulation and is inspired by TGN [20], which we adapt to NIDS. Through this approach we seek to improve the detection process by incorporating temporal information and representing it in a continuous manner, as opposed to discrete snapshots.

IV. Channel-Centric Spatio-Temporal Graph Network

We start with a high-level overview of the proposed architecture of CCSTGN. CCSTGN processes a sequence of traffic flows in batches of size B. Flows are mapped to a set $\mathcal{V}$ of communication channels, which are the nodes of the channel-centric graph representation described in Sec. III-B. Fig. 2 illustrates the four main components of the CCSTGN architecture. Numbered arrows correspond to the main processing steps in the architecture, which we describe below, while non-numbered arrows represent dependencies. These four main components are:

1) *Flow embedding.* The features $\mathbf{f}_i$ of all flows in a batch are embedded into a lower-dimensional space to obtain flow embeddings $\mathbf{f}_i^{emb}$.
2) *Node state update.* The node state of each node $v \in \mathcal{V}$ is updated based on the flow embeddings $\mathbf{f}_i^{emb}$ associated with it in the batch, retrieved from the embedding storage.
3) *Node state embedding.* The updated node state is accessed to generate a node state embedding $\mathbf{f}_v^{mem}$ for each of the observed nodes at the observed timestamps based on their corresponding spatio-temporal neighborhoods using a GNN.
4) *Embedding storage.* The flow embeddings $\mathbf{f}_i^{emb}$, flow timestamps t_i, and the node state embeddings $\mathbf{f}_v^{mem}$ together with their timestamps $t^{(k)}$, are stored for the subsequent node state memory update and for classification.

The output of CCSTGN are flow embeddings $\mathbf{f}_i^{emb}$ and node state embeddings $\mathbf{f}_v^{mem}$. The embeddings can be further processed in order to generate predictions for the considered task, which in our case is anomaly detection through binary classification.

A. Flow embedding

As a first step in the CCSTGN architecture, the feature vectors of the flows in batch k are projected to a lower-dimensional representation vector. The main reason behind this operation is to provide a more compact representation of the features. For flow i with features $\mathbf{f}_i$ and timestamp t_i we obtain the embedding $\mathbf{f}_i^{emb}$ using a two-layer neural network,

$$\mathbf{f}_i^{emb} = \sigma\left(\mathbf{f}_i \cdot \mathbf{W}_0 + \mathbf{b}_0\right) \cdot \mathbf{W}_1 + \mathbf{b}_1, \tag{1}$$

where σ is a non-linear activation function, $\mathbf{W}_0 \in \mathbb{R}^{N \times P}$ and $\mathbf{W}_1 \in \mathbb{R}^{P \times M}$ are learnable weight matrices, and $\mathbf{b}_0 \in \mathbb{R}^P$ and $\mathbf{b}_1 \in \mathbb{R}^M$ are learnable bias vectors. We use ReLU as the activation function and $P = (N + M)/2$. The computed flow embeddings are then stored in the embedding storage (see Section IV-D).

B. Node state update

Following TGN [20], we define a memory module that serves as a compact representation of each node's history. It is implemented as a matrix that contains an M-dimensional vector representation of the state $s_v \in \mathbb{R}^M$ for each node $v \in \mathcal{V}$ that the model has seen so far, associated with a timestamp $t_v \in \mathbb{R}_{\geq 0}$, which corresponds to the time stamp of the most recent flow in previous batches that triggered an update of the node's state. The node state for a previously unseen node v is initialized to a zero vector and its timestamp is $t_v = 0$.

Due to batching flows, node v can have multiple embeddings associated with it in the embedding storage. Let us denote by $Z(v) = \{i : v_i = v\}$ the set of flow embeddings associated with node v in the embedding storage. We aggregate those embeddings by computing

$$\mathbf{f}_v^{agg} = \frac{1}{|Z(v)|} \sum_{i \in Z(v)} \mathbf{f}_i^{emb}, \tag{2}$$

where $\mathbf{f}_v^{agg}$ is the aggregated embedding for node v. We initially explored more complex aggregation strategies that also consider the stored memory embeddings and timestamps, but preliminary evaluations showed that this simple strategy was sufficient for achieving good performance.

After aggregating the flow embeddings in batch k, the state for nodes v with associated flows (i.e., $|Z(v)| > 0$) is updated as

$$\mathbf{s}_v^{(k)} = \text{RNN}\left(\mathbf{s}_v^{(k-1)}, \mathbf{f}_v^{agg}\right), \tag{3}$$

where $\mathbf{s}_v^{(k)}$ is the updated state for node v, and $\mathbf{s}_v^{(k-1)}$ is the previous state for node v. The new node state is calculated by applying an RNN, with the previous state for that node as its hidden state, to the newly computed aggregation of flow embeddings. We use a GRU [17] as the RNN in our implementation.

We then update the timestamp associated with the last update of the state of node v to the timestamp of the most recent flow associated with node v, *i.e.,*

$$t_v^{(k)} = \max_{i \in Z(v)} t_i.$$

The node state and the timestamp are only updated for the nodes observed in the current input batch (*i.e.,* $\exists i$ s.t. $v_i = v$ for flow i), we set $\mathbf{s}_v^{(k)} = \mathbf{s}_v^{(k-1)}$ and $t_v^{(k)} = t_v^{(k-1)}$ for the other nodes. After the update of the node state $\mathbf{s}_v^{(k)}$ and its timestamp $t_v^{(k)}$, the embedding storage for node v is emptied. Finally, it is also important to note that the node state is continuously updated during the testing phase, even after training has concluded.

C. Node state embedding

Next, we introduce two alternative GNN models to be used in order to generate the node state embeddings f_v^{mem}. Before describing the GNN models, we introduce our definition of the neighborhood of a node. Given that we are using communication channels as the nodes of our graph, the nodes are identified both by their node ID $v \in \mathcal{V}$, and by the unordered pair of IP addresses $\{\text{IP}_{v,1}, \text{IP}_{v,2}\}$ associated with their incident host devices. In particular, we define the bijective mapping $\varphi(v) = \{\text{IP}_{v,1}, \text{IP}_{v,2}\}$, which is computed during the data preprocessing stage. At time $t^{(k)} = \max_{i<Bk} t_i$ (i.e., upon processing batch k) we can then define the spatio-temporal neighborhood of node $v \in \mathcal{V}$ as

$$\mathcal{N}_{t^{(k)}}(v) := \{v' \in \mathcal{V} : \varphi(v') \cap \varphi(v) \neq \emptyset \wedge t_v \leq t_{v'} \leq t^{(k)}\}, \tag{4}$$

where $t_v, t_{v'} \in \mathbb{R}$ correspond to the last update times of the node state for nodes v and v'. The spatio-temporal neighborhood of node v is thus defined as the set of nodes that have at least one associated IP address in common and whose node state has been updated no later than the node state of node v. Moreover, we also define the spatio-temporal degree $d_{t^{(k)}}(v) = |\mathcal{N}_{t^{(k)}}(v)|$ of node v at time $t^{(k)}$, *i.e.,* the cardinality of its spatio-temporal neighborhood at time $t^{(k)}$.

Finally, to limit the memory and computing requirements of our solution, we select the η most recently updated neighbors (largest timestamp values) from $\mathcal{N}_{t^{(k)}}(v)$ together with the node itself. That is, we construct a sampled spatio-temporal neighborhood $\mathcal{N}'_{t^{(k)}}(v)$ for node $v \in \mathcal{V}$ at time $t^{(k)}$ as

$$\mathcal{N}'_{t^{(k)}}(v) := \{v\} \cup \operatorname*{argmax}_{\mathcal{N}' \subseteq \mathcal{N}_{t^{(k)}}(v) \setminus \{v\}, |\mathcal{N}'| = \eta} \sum_{v' \in \mathcal{N}'} t_{v'}. \tag{5}$$

1) CCSTGN-rs: The first GNN approach defines the first model instance of our architecture, namely CCSTGN - residual symmetric (CCSTGN-rs), and is inspired by GCN$_{gf}$ [23], which is designed to be robust against heterophily in graphs, potentially avoiding the problem of *over-smoothing*.

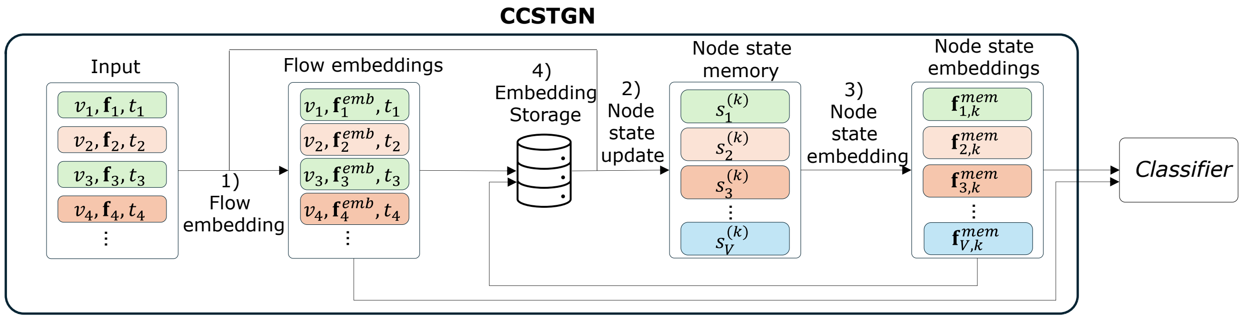

Fig. 2. High-level overview of the CCSTGN architecture. Numbered edges represent processing steps, non-numbered edges are dependencies. Different colors represent different nodes. A node state s_v is kept in the memory for each node. It is updated based on the aggregated embeddings of the flows associated to the node in a batch of flows. The node states are processed using a GNN to obtain a node state embedding, which is used for classifying new flows based on their embedding. The architecture supports various training approaches (supervised, unsupervised) for the classifier (Section V-C).

Following this approach, for node v we compute the node state embedding $\mathbf{f}_{v,k}^{mem} = \mathbf{f}_{v,k}(L)$ based on batch k as

$$\begin{aligned}\mathbf{f}_{v,k}(0) &= \mathbf{s}_v^{(k)}\\ \mathbf{f}_{v,k}(l{+}1) &= \mathbf{f}_{v,k}(l)\\ &\quad+\sigma\left(\sum_{v'\in\mathcal{N}'_{t^{(k)}}(v)} \frac{1}{\sqrt{d_{t^{(k)}}(v)d_{t^{(k)}}(v')}}\cdot\mathbf{f}_{v',k}(l)\cdot\mathbf{W}_S\right),\end{aligned} \tag{6}$$

where $0 \le l < L$ is the layer of the model, σ is a non-linear activation function (*e.g.,* ReLU), and $\mathbf{W}_S$ is defined as

$$\mathbf{W}_S := \frac{\mathbf{W}+\mathbf{W}^T}{2},$$

with $\mathbf{W} \in \mathbb{R}^{M\times M}$ being a learnable weight matrix shared across layers.

2) CCSTGN-iso: The second GNN approach that we consider is inspired by GIN [24], and it defines the second model instance of our architecture: CCSTGN - isomorphism (CCSTGN-iso). We expect this model to be faster than the CCSTGN-rs model, especially for a high number of layers, since this approach does not require the computation of the spatio-temporal degree of the sampled neighbors.

Using this approach, for node $v \in \mathcal{V}$ we compute the node state embedding $f_{v,k}^{mem} = f_{v,k}(L)$ based on batch k as

$$\begin{aligned}\mathbf{f}_{v,k}(0) &= \mathbf{s}_v^{(k)}\\ \mathbf{f}_{v,k}(l+1) &= \mathrm{MLP}^{(l+1)}\Bigg(\left(1+\epsilon^{(l+1)}\right)\cdot\mathbf{f}_{v,k}(l)+\\ &\qquad\sum_{v'\in\mathcal{N}'_{t^{(k)}}(v)} \mathbb{1}_{[v'\neq v]}\,\mathbf{f}_{v',k}(l)\Bigg),\end{aligned} \tag{7}$$

where $0 \le l < L$ is the layer of the model, $\epsilon^{(l+1)}$ is a learnable parameter associated with layer $l+1$ and we define $\mathrm{MLP}^{(l+1)}$ for layer $l+1$ as a two-layer neural network:

$$\mathbf{f}_{v,k}(l{+}1) = \sigma\left(\mathbf{f}_{v,k}^{sum}(l)\cdot\mathbf{W}_0^{(l+1)}+\mathbf{b}_0^{(l+1)}\right)\cdot\mathbf{W}_1^{(l+1)}+\mathbf{b}_1^{(l+1)}$$

where σ is a non-linearity, $\mathbf{f}_{v,k}^{sum} \in \mathbb{R}^M$ is the result of the input to $\mathrm{MLP}^{(l+1)}$ (which is computed as shown in Eqn. (7)), $\mathbf{W}_0^{(l+1)}, \mathbf{W}_1^{(l+1)} \in \mathbb{R}^{M\times M}$ are learnable weight matrices, and $\mathbf{b}_0^{(l+1)}, \mathbf{b}_1^{(l+1)} \in \mathbb{R}^M$ are learnable bias vectors.

D. Embedding storage

Inspired by TGN [20], the embedding storage stores embeddings organized by the node they belong to. It stores the flow embeddings f_i^{emb} along with their timestamp t_i and the node $v_i \in \mathcal{V}$ they are associated with. In addition, for each node $v \in \mathcal{V}$ it may store the most recent node state embedding $\mathbf{f}_{v,k}^{mem}$ together with its timestamp $t^{(k)}$ (defined in Section IV-C), for the purpose of classification (as explained in Section V-C). The embedding storage can be implemented as a dictionary (key-value store), where the key is the node ID and the values are lists of associated embeddings and timestamps, which are initially empty.

V. Evaluation Methodology

A. Data

In order to evaluate our proposed CCSTGN models, we use the publicly available NF-UNSW-NB15-v2 NIDS dataset [25]. We choose this dataset due to an existing performance evaluation in [13] of multiple NIDS approaches for this dataset, which allows for easier comparison between those models and CCSTGN. NF-UNSW-NB15-v2 consists of $2,390,275$ flows, each having 43 features, of which 3.98% flows are labeled as attacks and 96.02% as benign. A category is also specified for each sample labeled as an attack.

B. Data preprocessing

1) Node ID mapping: We first define the bijective mapping $\varphi : \mathcal{V} \to \{\mathrm{IP}, \mathrm{IP}\}$ for the dataset, which is used for defining the spatio-temporal neighborhood for the computation of memory embeddings. We construct φ by its inverse φ^{-1}, a mapping from unordered IP pairs to node IDs. We create φ^{-1} by iterating over the flows in the dataset and assigning an incremental count of previously encountered IP pairs as the corresponding node ID v if the pair is new, skipping already encountered pairs.

2) Timestamp considerations: The NF-UNSW-NB15-v2 dataset does not contain a timestamp feature, so we assume that the flows are arranged in sequential order and evenly spaced in time, assigning timestamp $t_i = i$ to flow i $\forall i \in [0, \ldots, D-1]$, where D is the number of flows in the dataset. If the dataset were to include timestamps as a feature of the flows, this step would consist of sorting the samples in the dataset by their timestamps to feed them to our model in a sequential manner.

3) Training-validation-test split: We then split the dataset into training, validation, and test sets with a $70-10-20$ split without shuffling to maintain their temporal order. The train and validation sets are combined and used to train our models in experiments VI-D and VI-E.

4) Feature preprocessing: The flow features are composed of both numerical and categorical features. We standardize each numerical feature using their respective means and standard deviations in the training set. We use a binary encoder to encode the categorical features, into a binary representation with a set number of bits. The number of bits is set to be able to encode the entire range of possible values defined in the NetFlow specification [26]. This potentially oversizes the model in regards to the data, as not all values may appear in the samples, but it ensures the applicability of our approach in real-world scenarios, and avoids data leakage caused by setting the number of bits based on the entire dataset. We note that the encoding scheme of categorical features tends to not be described in detail throughout the NIDS approaches discussed in Section II, which limits reproducibility if the source code is not available.

C. Training strategies

We explore the effect of applying various training strategies to our proposed CCSTGN architecture. We describe the training strategies for K batches of flows of batch size B.

1) Supervised learning: Firstly, we use the labels in the dataset and train the models in a supervised manner. For this, we connect the outputs (node state embeddings and flow embeddings) of the CCSTGN model to a classification head, which is trained together with the CCSTGN model. We use a two-layer neural network as classification head, whose inputs are a flow embedding and the most recent memory embedding of its associated node, i.e.,

$$\hat{\mathbf{y}}_i = softmax\left(\sigma\left(\left(\mathbf{f}_i^{emb} \| \mathbf{f}_{v_i,k}^{mem}\right) \cdot \mathbf{W}_0 + \mathbf{b}_0\right) \cdot \mathbf{W}_1 + \mathbf{b}_1\right),$$

where σ is a non-linearity, $\|$ is the concatenation operation, $\mathbf{W}_0 \in \mathbb{R}^{2M \times Q}$ and $\mathbf{W}_1 \in \mathbb{R}^{Q \times C}$ are learnable weight matrices, and $\mathbf{b}_0 \in \mathbb{R}^Q$ and $\mathbf{b}_1 \in \mathbb{R}^C$ are learnable bias vectors. We use $Q = (2M + C)/2$, where C is the number of classes of the considered problem.

We choose to find the optimal set of parameters $\boldsymbol{\theta}^*$ for our model and for the classification head by minimizing a weighted cross-entropy loss function, due to its robustness to imbalanced class distributions:

$$\begin{aligned}\boldsymbol{\theta}^* &= \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, \mathcal{L}_{WCE}(\mathbf{y}, \hat{\mathbf{y}}; \boldsymbol{\theta}) \\ &= \underset{\boldsymbol{\theta}}{\operatorname{argmin}} -\frac{1}{KB} \sum_{k=1}^{K} \sum_{i=(k-1)B}^{kB-1} \sum_{c=0}^{C-1} w_c y_i^c \log(\hat{y}_i^c),\end{aligned}$$

where K is the number of batches, B is the batch size, C is the number of classes, y_i^c is the true probability (the labels are one-hot encoded) of class c for flow i (in batch k), $\hat{y}_i^c$ is the predicted probability of class c for flow i (in batch k), and w_c is the weight associated with class c, computed as

$$w_c = \frac{s}{C \cdot s_c} \quad \forall c \in \{0, \ldots, C-1\},$$

where s denotes the total number of samples in the training set, and s_c denotes the number of samples in the training set that correspond to class c.

2) Self-supervised learning: The second training strategy for our CCSTGN models does not consider the available labels and is instead based on self-supervised learning.

We define the learning objective in this case as maximizing the similarity between the generated flow embeddings $\mathbf{f}^{emb}$ and the memory embeddings $\mathbf{f}^{mem}$ according to the cosine similarity, since the cosine similarity ignores the magnitude difference between the embeddings, and instead considers only their angles. We thus obtain the optimal parameters as

$$\begin{aligned}\boldsymbol{\theta}_m^* &= \underset{\boldsymbol{\theta}_m}{\operatorname{argmin}}\, \mathcal{L}_{cos}(\mathbf{f}^{emb}, \mathbf{f}^{mem}; \boldsymbol{\theta}_m) \\ &= \underset{\boldsymbol{\theta}_m}{\operatorname{argmin}} -\frac{1}{KB} \sum_{k=1}^{K} \sum_{i=(k-1)B}^{kB-1} 1 - \cos(\mathbf{f}_i^{emb}, \mathbf{f}_{v_i,k}^{mem}).\end{aligned}$$

Lastly, we choose to define the class probability distributions associated with the obtained embeddings for the specific case of binary classification as

$$\hat{\mathbf{y}}_i = softmax\left(\cos(\mathbf{f}_i^{emb}, \mathbf{f}_{v_i,k}^{mem}), -\cos(\mathbf{f}_i^{emb}, \mathbf{f}_{v_i,k}^{mem})\right).$$

Note that in the case of both training strategies, the flow embedding of flow i is compared to the node state embedding obtained based on the batch it belongs to. An alternative could be to compare the flow embedding to the previous node state embedding, but we did not explore this alternative.

3) Fine-tuning: An additional training strategy we explored in our experiments is fine-tuning CCSTGN models trained using self-supervised learning by then training them with supervised learning. In that sense, we once more add the classification head defined in Section V-C1 and train the models using the weighted cross-entropy loss function defined in that section. Thus, this training strategy follows the same considerations and definitions from the supervised learning approach, but uses a non-random initialization of the parameters of the CCSTGN models, instead harnessing the final parameter configuration of the pre-trained self-supervised models.

D. *Evaluation metrics*

In order to provide a comprehensive analysis of the performance of the considered models, we apply multiple evaluation metrics in our experiments.

First, we report the Area Under the ROC Curve (AUC), which is an aggregate measure of the performance of the model across different classification thresholds, thus providing a global evaluation of the classifier, taking into account all possible trade-offs between the True Positive Rate (TPR) and the False Positive Rate (FPR).

Second, we report the FPR, the False Negative Rate (FNR), and the F1-score, which expect class label predictions instead of class probability distributions. For the first experiment, we follow a strategy similar to that defined in EULER [15], where we identify an optimal cut-off threshold for classification ω, given a balance between FPR and FNR controlled using the parameter λ. The threshold is obtained as the solution to the following optimization problem

$$\omega = \underset{\omega'}{\operatorname{argmin}}\, F(\omega') = \underset{\omega'}{\operatorname{argmin}}\, \lambda \ \mathrm{FNR}(\omega') + (1-\lambda)\ \mathrm{FPR}(\omega').$$

For the sake of our experiments, we set $\lambda = 0.75$, and the calculate the threshold ω using the validation set. The threshold is then used for calculating the reported evaluation results for the test set. Lastly, we also report the $F(\omega)$ value corresponding to the identified optimal threshold. For the second experiment, the threshold is set based on a set of fixed FPR values.

VI. Numerical Results

A. *Experimental details*

In all of the described experiments, the models were trained and evaluated on a laptop with an Intel i7-8750H (12) @ 4.100GHz CPU, 16GB DDR4, and an NVIDIA GeForce GTX 1060 Mobile GPU. Both the training and evaluation were executed on the GPU using CUDA version 12.1. We used the Adam optimizer [27] with a learning rate of 0.01, and kept the default values for the remaining hyperparameters, since preliminary explorations of their values did not show significant impact on the evaluation metrics.

B. *Baselines*

We distinguish between two different groups of baselines. One group is defined by GNN-based approaches for NIDS, whose performance on the NF-UNSW-NB15-v2 dataset was evaluated and reported in [13]: TGN [20], GAT [28], E-GraphSAGE [12], EULER [15], and 3D-IDS [13]. This selection of models contains representatives of both general purpose GNN models (TGN [20] and GAT [28]), and GNN-based models designed for NIDS (E-GraphSAGE [12], EULER [15], and 3D-IDS [13]). Moreover, it also includes representatives of static GNN-based models (GAT [28] and E-GraphSAGE [12]), and dynamic GNN-based models (TGN [20], EULER [15], and 3D-IDS [13]).

We also include two baseline methods that are not GNN-based and that we implement and evaluate according to the same experimental configuration and data preprocessing that we used for our proposed CCSTGN models. We first consider a Multinomial Logistic Regression (MLR) model:

$$\hat{\mathbf{y}}_{v,t} = softmax\left(\mathbf{f}_{v,t} \cdot \mathbf{W} + \mathbf{b}\right),$$

where $\mathbf{f}_{v,t} \in \mathbb{R}^N$ is the input feature vector associated with an input sample with node ID v and timestamp t, $\mathbf{W} \in \mathbb{R}^{N \times C}$ is a learnable weight matrix, $\mathbf{b} \in \mathbb{R}^C$ is a learnable bias vector.

Next, we also define a Multi-Layer Perceptron (MLP) model as a 4-layer neural network:

$$\begin{aligned}
\mathbf{h}_{v,t} &= \sigma\left(\sigma\left(\mathbf{f}_{v,t} \cdot \mathbf{W}_0 + \mathbf{b}_0\right) \cdot \mathbf{W}_1 + \mathbf{b}_1\right)\\
\hat{\mathbf{y}}_{v,t} &= softmax\left(\sigma\left(\mathbf{h}_{v,t} \cdot \mathbf{W}_2 + \mathbf{b}_2\right) \cdot \mathbf{W}_3 + \mathbf{b}_3\right),
\end{aligned}$$

where $\mathbf{f}_{v,t} \in \mathbb{R}^N$ is the input feature vector associated with an input sample with node ID v and timestamp t, σ is a ReLU non-linearity, $\mathbf{W}_0 \in \mathbb{R}^{N \times P}$, $\mathbf{W}_1 \in \mathbb{R}^{P \times Q}$, $\mathbf{W}_2 \in \mathbb{R}^{Q \times M}$, $\mathbf{W}_3 \in \mathbb{R}^{M \times C}$ are learnable weight matrices, and $\mathbf{b}_0 \in \mathbb{R}^P$, $\mathbf{b}_1 \in \mathbb{R}^Q$, $\mathbf{b}_2 \in \mathbb{R}^M$, and $\mathbf{b}_3 \in \mathbb{R}^C$ are learnable bias vectors. We set M to the same dimensionality as the memory embedding used by the CCSTGN models, C the number of classes for the classification, and we set $P = (N + M)/2$ and $Q = (M + C)/2$.

C. *Hyperparameter exploration*

We performed a limited hyperparameter exploration that only included the supervised learning strategy. Trends in model performance on the validation set for different values of relevant hyperparameters were recorded, and used for manual parameter selection. Based on the study, we define a hyperparameter configuration consisting of a batch size of 10 000 flows, 7 epochs, 1-layer GNN models for the computation of the memory embeddings, and a memory dimensionality of $M = 20$. The CCSTGN - residual symmetric (CCSTGN-rs) models are defined to sample $\eta = 7$ spatio-temporal neighbors and the CCSTGN - isomorphism (CCSTGN-iso) models sample $\eta = 5$ neighbors.

D. *Binary classification performance*

In this experiment, we evaluate the performance of our proposed models (following all the aforementioned training strategies) and compare the results to those of the baselines described in Section VI-B for the task of binary classification using the NF-UNSW-NB15-v2 dataset. Table I shows the classification results, together with the average time required for the training and for the testing of each model on the complete training and test datasets, respectively.

As a first observation from those results, we note that for the self-supervised approach the models perform relatively well, especially in terms of the AUC metric, considering that they do not use labels during the training procedure. Thus, the results lead us to believe that this is a valid training strategy for our proposed architecture, which deserves further exploration in the context of appropriate self-supervised tasks. Furthermore, we observe that fine-tuning provides higher performance than the self-supervised approach, and a slight improvement for

TABLE I
BINARY CLASSIFICATION RESULTS CORRESPONDING TO THE MEAN AND STANDARD DEVIATION AFTER 5 RUNS FOR EACH METHOD. THE VALUES ARE PERCENTAGES. THE RESULTS WITH † ARE DIRECTLY COPIED FROM [13]. **First**, $\overline{\textbf{Second}}$, $\underline{\textbf{Third}}$ HIGHEST MEAN VALUE PER METRIC. TRAINING STRATEGIES OTHER THAN SUPERVISED LEARNING ARE REPRESENTED AS (SSL) FOR SELF-SUPERVISED LEARNING, AND (FT) FOR FINE-TUNING.

Method	AUC ↑	F1 ↑	FPR ↓	FNR ↓	$F(\omega)$ ↓	Training time	Test time
GAT [28]†	89.91 ± 0.79	92.20 ± 1.26	-	-	-	-	-
TGN [20]†	88.01 ± 1.40	93.55 ± 0.48	-	-	-	-	-
E-GraphSAGE [12]†	90.39 ± 0.51	94.10 ± 0.57	-	-	-	-	-
EULER [15]†	86.97 ± 1.05	92.76 ± 0.93	-	-	-	-	-
3D-IDS [13]†	91.55 ± 1.01	$\mathbf{95.45\pm0.82}$	-	-	-	-	-
MLR	99.522 ± 0.004	89.604 ± 0.105	1.793 ± 0.021	0.319 ± 0.003	0.688 ± 0.004	$34m22s$	$01m23s$
MLP	$\mathbf{99.915\pm0.004}$	$\overline{\mathbf{95.224}\pm0.107}$	$\mathbf{0.785\pm0.019}$	$\underline{\mathbf{0.046}\pm0.014}$	$\mathbf{0.231\pm0.009}$	$34m08s$	$01m22s$
CCSTGN-rs	99.881 ± 0.027	94.618 ± 0.801	0.894 ± 0.142	$\mathbf{0.022\pm0.006}$	$\underline{\mathbf{0.240}\pm0.032}$	$39m38s$	$01m40s$
CCSTGN-iso	$\overline{\mathbf{99.886}\pm0.027}$	94.969 ± 0.267	0.829 ± 0.045	$\overline{\mathbf{0.041}\pm0.026}$	$\overline{\mathbf{0.238}\pm0.027}$	$37m09s$	$01m31s$
CCSTGN-rs (ssl)	81.849 ± 3.508	31.333 ± 7.829	31.900 ± 17.072	16.547 ± 10.900	20.385 ± 4.540	$39m27s$	$01m39s$
CCSTGN-iso (ssl)	88.997 ± 2.946	29.892 ± 8.535	38.705 ± 18.358	7.565 ± 5.017	15.350 ± 2.685	$37m04s$	$01m31s$
CCSTGN-rs (ft)	$\underline{\mathbf{99.884}\pm0.012}$	95.037 ± 0.445	$\underline{\mathbf{0.816}\pm0.076}$	0.057 ± 0.018	0.247 ± 0.031	$39m41s$	$01m41s$
CCSTGN-iso (ft)	99.882 ± 0.011	$\underline{\mathbf{95.058}\pm0.424}$	$\overline{\mathbf{0.796}\pm0.051}$	0.251 ± 0.275	0.388 ± 0.218	$38m19s$	$01m35s$

certain metrics over the performance of the supervised models, especially for CCSTGN-rs.

In addition, we can see that both our supervised and fine-tuned models provide similar performance, and both of them outperform all the baseline models evaluated in [13], except, arguably, from 3D-IDS [13] in terms of the F1-score. We believe that these results indicate that our proposed approach (*i.e.,* channel-centric graph representation of the network and continuous temporal graph representation) indeed improves the performance for NIDS over existing GNN-based approaches in the NF-UNSW-NB15-v2 dataset. Surprisingly, we should remark that our models' training setup differs from that of the approaches evaluated in [13] in that our models are trained for much fewer epochs (7 as opposed to 500), without explicit regularization or learning rate scheduling mechanisms, and with a very slightly optimized hyperparameter configuration.

However, we should note the high performance of the MLP model seems to indicate that in this dataset, the architectural complexity of our proposed CCSTGN approaches and of the other GNN-based baselines does not yield a noticeable benefit for the classification results, and that, instead, a careful data preprocessing strategy can potentially affect the performance in a more significant manner. Nevertheless, the MLR model shows worse performance than the MLP, indicating that certain complexity is needed for the classification of the input features into their corresponding classes.

Comparing the training times, we can conclude that the additional computational complexity of CCSTGN compared to the MLP is not significant. We can make the same observation about inference, as all models are able to classify a batch of 10000 flows in approximately 2 seconds (c.f., the number of batches in the test set).

E. Missed Detections under FPR Constraints

In our second experiment, we analyze the miss rate (FNR) of the models at fixed false positive rates. This allows us to study the applicability of our models in real-world NIDS scenarios, where a limit on the false positive rate may be demanded.

We first identified the classification thresholds for each model that result in FPR values that are lower and closest to a set of predefined percentages, $\{0.01\%, 0.1\%, 1.0\%\}$, calculated using the validation set. We then used those thresholds on the test set and report the resulting metrics. Table II shows the resulting F1-score and FNR for each model. These results illustrate the performance of the different models when they are tuned for a specific FPR, which is an important constraint for practical use.

Analyzing the obtained results, we observe that the baseline MLP model shows the best overall performance, while the MLR and self-supervised CCSTGN models perform the worst. In addition, we observe that the supervised and fine-tuned CCSTGN models perform better when higher FPR percentages are allowed, displaying the lowest FNR score for FPR$_{val} \approx 1.0\%$. These results are consistent with those presented in VI-D, since Table I showed that the MLP provides the lowest FPR value, while the supervised CCSTGN models provide the lowest FNR values.

VII. CONCLUSION

In this work, we present a novel architecture for NIDS, namely CCSTGN, which results from a novel combination of graph and temporal representations designed for NIDS that we believe address issues in existing GNN-based NIDS approaches. The architecture is flexible in that it can be trained using multiple learning strategies. Our results show that CCSTGN is able to outperform the results of several GNN-based NIDS approaches reported for the NF-UNSW-NB15-v2 dataset [13]. However, we also show that a simple MLP model provides very competitive and arguably better performance than both CCSTGN and the GNN-based alternatives, which we conjecture is the result of our data preprocessing strategy. There are many interesting directions for future research that could potentially extend our results. These include exploring alternative graph and temporal representations to the ones considered in the definition of CCSTGN in the context of NIDS, including graph attention networks. Of significant practical

TABLE II

MISSED DETECTIONS UNDER FPR CONSTRAINTS RESULTS CORRESPONDING TO THE MEAN AND STANDARD DEVIATION AFTER 5 RUNS FOR EACH METHOD. **Highest** MEAN VALUE PER METRIC. ALL VALUES ARE EXPRESSED AS PERCENTAGES. TRAINING STRATEGIES OTHER THAN SUPERVISED LEARNING ARE REPRESENTED AS (SSL) FOR SELF-SUPERVISED LEARNING, AND (FT) FOR FINE-TUNING.

Method	FPR$_{val}$ $\approx 0.01\%$		FPR$_{val}$ $\approx 0.1\%$		FPR$_{val}$ $\approx 1.0\%$	
	F1 $\uparrow$	FNR $\downarrow$	F1 $\uparrow$	FNR $\downarrow$	F1 $\uparrow$	FNR $\downarrow$
MLR	3.058 ± 0.318	98.446 ± 0.164	12.971 ± 0.741	93.037 ± 0.426	88.846 ± 0.142	11.401 ± 0.228
MLP	$\mathbf{65.829}\pm\mathbf{2.234}$	$\mathbf{50.770}\pm\mathbf{2.489}$	$\mathbf{82.295}\pm\mathbf{1.622}$	$\mathbf{29.501}\pm\mathbf{2.444}$	$\mathbf{95.159}\pm\mathbf{0.119}$	0.038 ± 0.009
CCSTGN-rs	50.331 ± 7.543	65.988 ± 6.425	77.020 ± 4.533	36.579 ± 6.030	93.667 ± 0.857	$\mathbf{0.007}\pm\mathbf{0.003}$
CCSTGN-iso	22.321 ± 10.588	87.036 ± 6.617	60.763 ± 3.777	56.106 ± 3.904	94.773 ± 0.309	0.061 ± 0.085
CCSTGN-rs (ssl)	1.026 ± 1.665	99.191 ± 1.321	1.216 ± 1.709	99.039 ± 1.360	1.632 ± 2.027	98.703 ± 1.616
CCSTGN-iso (ssl)	0.778 ± 0.090	99.609 ± 0.046	4.421 ± 1.115	97.699 ± 0.603	28.365 ± 10.045	78.620 ± 10.343
CCSTGN-rs (ft)	49.784 ± 5.229	66.585 ± 4.631	70.193 ± 4.563	45.329 ± 5.604	95.029 ± 0.582	0.080 ± 0.021
CCSTGN-iso (ft)	46.941 ± 11.552	68.555 ± 9.807	70.124 ± 8.713	44.946 ± 11.109	94.934 ± 0.329	0.044 ± 0.041

interest would be exploring non-binary classification performance, to provide more fine-grained information to security analysts. Another interesting direction is analyzing the effect of data preprocessing on the performance of NIDS models, to assess whether data preprocessing has a greater impact on the performance than the addition of GNN components.

ACKNOWLEDGMENT

The work was partly funded by Digital Futures through the CLAIRE project, the Swedish Civil Contingencies Agency (MSB) through the CERCES2 project and by the Swedish Research Council through project 2020-03860.

REFERENCES

[1] I. Lella, C. Ciobanu, M. Theocharidou, E. Magonara, A. Malatras, R. Svetozarov Naydenov, and E. Tsekmezoglou. *ENISA threat landscape 2023 : July 2022 to June 2023*. European Union Agency for Cybersecurity, 2023.

[2] S Latha and Sinthu Janita Prakash. A survey on network attacks and intrusion detection systems. In *Proc. of IEEE International Conference on Advanced Computing and Communication Systems*, pages 1–7, 2017.

[3] Matt Weir, Sudhir Aggarwal, Breno De Medeiros, and Bill Glodek. Password cracking using probabilistic context-free grammars. In *Proc. of IEEE Symposium on Security and Privacy*, pages 391–405, 2009.

[4] Ross Anderson. *Security engineering: a guide to building dependable distributed systems*. John Wiley & Sons, 2020.

[5] Felix Erlacher and Falko Dressler. FIXIDS: a high-speed signature-based flow intrusion detection system. In *Proc. of IEEE/IFIP Network Operations and Management Symposium*, pages 1–8, 2018.

[6] Philokypros Ioulianou, Vasileios Vasilakis, Ioannis Moscholios, and Michael Logothetis. A signature-based intrusion detection system for the Internet of Things. *Information and Communication Technology Form*, 2018.

[7] A.A. Waskita, H. Suhartanto, P.D. Persadha, and L.T. Handoko. A simple statistical analysis approach for intrusion detection system. In *Proc. of Conference on Systems, Process & Control*, pages 1–8, 2013.

[8] Roberto Doriguzzi-Corin, Stuart Millar, Sandra Scott-Hayward, Jesus Martinez-del Rincon, and D. Siracusa. LUCID: A practical, lightweight deep learning solution for DDoS attack detection. *IEEE Transactions on Network and Service Management*, 17(2):876–889, 2020.

[9] Yeongwoo Kim and György Dán. An active learning approach to dynamic alert prioritization for real-time situational awareness. In *Proc. of IEEE Conference on Communications and Network Security (CNS)*, pages 154–162. IEEE, 2022.

[10] Yeongwoo Kim, György Dán, and Quanyan Zhu. Human-in-the-loop cyber intrusion detection using active learning. *IEEE Transactions on Information Forensics and Security*, pages 1–16, to appear, 2024.

[11] Siddharth Bhatia, Arjit Jain, Pan Li, Ritesh Kumar, and Bryan Hooi. Mstream: Fast anomaly detection in multi-aspect streams. In *Proc. of WWW*, 2021.

[12] Wai Weng Lo, Siamak Layeghy, Mohanad Sarhan, Marcus Gallagher, and Marius Portmann. E-graphsage: A graph neural network based intrusion detection system for IoT. In *Proc. of IEEE/IFIP Network Operations and Management Symposium*, pages 1–9, 2022.

[13] Chenyang Qiu, Yingsheng Geng, Junrui Lu, Kaida Chen, Shitong Zhu, Ya Su, Guoshun Nan, Can Zhang, Junsong Fu, Qimei Cui, et al. 3D-IDS: Doubly disentangled dynamic intrusion detection. In *Proc. of ACM Conference on Knowledge Discovery and Data Mining (KDD)*, page 1965–1977, 2023.

[14] Qi Chen, Yifei Wang, Yisen Wang, Jiansheng Yang, and Zhouchen Lin. Optimization-induced graph implicit nonlinear diffusion. In *Proc. of International Conference on Machine Learning*, volume 162, pages 3648–3661. PMLR, 17–23 Jul 2022.

[15] Isaiah J King and H Howie Huang. Euler: Detecting network lateral movement via scalable temporal link prediction. *ACM Transactions on Privacy and Security*, 26(3), 2023.

[16] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[17] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proc. of Conference on Empirical Methods in Natural Language Processing*, October 2014.

[18] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in Neural Information Processing Systems*, 30, 2017.

[19] Andrea Venturi, Dario Stabili, and Mirco Marchetti. Problem space structural adversarial attacks for network intrusion detection systems based on graph neural networks. *arXiv preprint arXiv:2403.11830*, 2024.

[20] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and M. Bronstein. Temporal graph networks for deep learning on dynamic graphs. *arXiv preprint arXiv:2006.10637*, 2020.

[21] Ting Zhu, Xiaohui Chen, Li Chen, Weidong Wang, and Guo Wei. GCLR: GNN-based cross layer optimization for multipath TCP by routing. *IEEE Access*, 8:17060–17070, 2020.

[22] Shuai Zhang, Bo Yin, Weiyi Zhang, and Yu Cheng. Topology aware deep learning for wireless network optimization. *IEEE Transactions on Wireless Communications*, 21(11):9791–9805, 2022.

[23] Francesco Di Giovanni, James Rowbottom, Benjamin Paul Chamberlain, Thomas Markovich, and Michael M Bronstein. Understanding convolution on graphs via energies. *Transactions on Machine Learning Research*, 2023.

[24] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *Proc. of Int. Conf. on Learning Representations (ICLR)*, 2019.

[25] Mohanad Sarhan, Siamak Layeghy, and Marius Portmann. Towards a standard feature set for network intrusion detection system datasets. *Mobile networks and applications*, pages 1–14, 2022.

[26] Benoit Claise. Cisco systems netflow services export version 9. Technical report, 2004.

[27] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proc. of Int. Conf. on Learning Representations (ICLR)*, May 2015.

[28] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, Yoshua Bengio, et al. Graph attention networks. In *Proc. of Int. Conf. on Learning Representations (ICLR)*, Apr. 2018.