Detecting Edge and Node Anomalies with Temporal GNNs

Detecting Edge and Node Anomalies with Temporal GNNs

Andrea Cavallo
Delft University of Technology
Delft, Netherlands
a.cavallo@tudelft.nl

Luca Gioacchini
Politecnico di Torino
Turin, Italy
luca.gioacchini@polito.it

Luca Vassio
DAUIN, Politecnico di Torino
Turin, Italy
luca.vassio@polito.it

Marco Mellia
Politecnico di Torino
Turin, Italy
marco.mellia@polito.it

Abstract

Computer and social networks can be effectively represented as complex temporal graphs where entities (nodes) keep interconnecting through various relationships (edges), forming evolving structures. Anomaly Detection (AD) in such networks consists of identifying patterns diverging from what is expected or normal. This task is fundamental for the detection of potential threats – e.g. suspicious connections (*edge AD*) or misbehaving entities (*node AD*), and challenging due to the lack of a common definition of anomaly. However, the literature is scarce about solutions to detect node anomalies on temporal graphs. This work addresses three challenges in AD as found in computer and social networks: fast-evolving graph structure, lack of ground truth, and simultaneous presence of anomalous nodes and edges. For this, we propose to use *temporal Graph Neural Networks (tGNNs)* coupled with specialised AD blocks trained in a self-supervised way. We also embed an attention mechanism providing interpretability to the decision process. We extensively validate the tGNNs on synthetic and real-world datasets showing that they successfully detect both node and edge anomalies simultaneously (≈0.9 of average AUC).

CCS Concepts

• **Computing methodologies → Machine learning**; • **Security and privacy → Intrusion/anomaly detection and malware mitigation**.

Keywords

Anomaly Detection, Graph Neural Networks, Dynamic Graphs, Communications Networks

ACM Reference Format:
Andrea Cavallo, Luca Gioacchini, Luca Vassio, and Marco Mellia. 2024. Detecting Edge and Node Anomalies with Temporal GNNs. In *Proceedings of the 3rd GNNet Workshop: Graph Neural Networking Workshop (GNNet '24), December 9–12, 2024, Los Angeles, CA, USA*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3694811.3697818

GNNet '24, December 9–12, 2024, Los Angeles, CA, USA
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1254-8/24/12
https://doi.org/10.1145/3694811.3697818

1 Introduction

Several problems in computer, social or generic communications networks can be framed as the identification of anomalous behaviour, i.e. a pattern that differs from common events in the network [1, 25]. For example, intrusion detection can rely on the unusual behaviour of an intruder compared to legitimate users; fault detection may identify network operation changes possibly caused by (partial) malfunctioning; bot detection on social networks can be based on anomalous communication patterns [27].

The behaviour of these networks can be modelled by dynamic graphs where various entities (nodes) establish connections (edges) that evolve over time. These graphs can then be used for Anomaly Detection (AD), which aims at finding unusual edges (e.g. unexpected connections) or nodes (e.g. network intruders, attack victims, botnet nodes, spammers) [15, 23, 27]. Despite the importance of both tasks, most of the AD approaches on temporal graphs mainly focus on edges [2]. However, in networks the ability to find anomalous nodes is also key, and trivial adaptations of edge AD methods to node AD (e.g. a node is anomalous if it generates an anomalous edge) fail to adapt to the heterogeneity of behaviours of nodes (e.g. client vs servers, influencers vs followers) and call for more flexible and general solutions. Moreover, the common lack of reliable ground truth makes this task particularly challenging.

Recent approaches applied Deep Learning and Graph Neural Networks (GNNs) to AD on dynamic graphs, demonstrating their capability to model the standard network behaviour and identify entities that diverge from it [23]. These works mainly focus on edge AD, inject artificial anomalies and evaluate their approaches on various real-world graphs, some of which have much slower evolution than communications networks (e.g. citation networks).

In this work, we address these shortcomings by (i) finding anomalies in fast-evolving graphs, with (ii) no ground truth during training, (iii) focusing on both edge and node AD. We employ a self-supervised data-driven strategy and we test on synthetic graphs and real communications networks with injected or real anomalies.

Specifically, we model communication networks as temporal graphs and we use *temporal GNNs* (tGNNs) given their effectiveness in obtaining meaningful embeddings for nodes and edges [9, 14], which we then exploit to detect unusual behaviour. For this, we design two AD blocks based on Neural Networks (AD-NN): the first one identifies anomalous edges, the second one anomalous nodes. We train the overall NN (tGNN+AD-NN) end-to-end on a self-supervised task: given past observations, the NN learns the normal behaviour of the network by predicting its evolution. Since this training process only relies on available observations of the network, there is no need for anomalous labels during training. Once trained, we use the NN on new instances of the graph to label as anomalous those edges and nodes that diverge from the learned patterns.

We test the tGNNs on two synthetic and six real datasets with both injected and real anomalies. Overall, we show that the proposed NNs are able to detect both edge and node anomalies with high accuracy – AUC of ≈ 0.9 on real graphs, generally improving over baselines. Importantly, their performance remains excellent even in scenarios where the training data includes unknown anomalies as it happens in practice. Our code is available at https://github.com/SmartData-Polito/tgnn-for-node-edge-AD.

2 Preliminaries

Notation: We define a dynamic graph $\{\mathcal{G}^t\}_{t=1}^N$ as a sequence of N static undirected graphs $\mathcal{G}^t = (\mathcal{V}^t, \mathcal{E}^t)$, where $\mathcal{V}^t$ and $\mathcal{E}^t$ are, respectively, the set of nodes and edges at snapshot t. More specifically, an edge $\epsilon = (u, v, w) \in \mathcal{E}^t$ indicates a connection between nodes $u, v \in \mathcal{V}^t$ with weight $w \in \mathbb{R}^+$. Each node $u \in \mathcal{V}^t$ has a feature vector $x_u^t \in \mathbb{R}^{e_f}$, where e_f is the feature size, and vectors x_u^t form the rows of the feature matrix X^t. $\mathcal{N}(u)^t$ denotes the neighbours of node u at snapshot t and $\mathcal{N}'(u)^t = \mathcal{N}(u)^t \cup \{u\}$ is the extended neighbourhood that includes the node u. An edge ϵ and a node u at snapshot t have label $y_\epsilon^t, y_u^t \in \{0, 1\}$, where 0 stands for *normal* and 1 for *anomalous*. Since the training is self-supervised, we use the available labels only for validation and test.

Temporal Graph Neural Networks: In this work, we use the Graph Convolutional Network (GCN) model [17]. A GCN with L hidden layers computes, at each layer $l \in [0, L+1]$ and for each node u, the embedding $z_u^l \in \mathbb{R}^{e^l}$, where e^l is the size at layer l (for simplicity, we omit the time index t), by transforming and averaging the embeddings of the neighbours and the target node at the previous layer. Formally, $z_u^l = \sum_{v \in \mathcal{N}'(u)} \hat{w}_{uv} W^l z_v^{l-1}$, where $W^l \in \mathbb{R}^{e^l \times e^{l-1}}$ is a learnable weight matrix and $\hat{w}_{uv}$ is the weight of the edge between nodes u and v, normalised such that $\sum_{v \in \mathcal{N}'(u)} \hat{w}_{uv} = 1$. At the first layer, node embeddings are their features, i.e. $z_u^0 = x_u$ and $e^0 = e_f$. Since dynamic graphs evolve temporally and spatially, *temporal GNNs* need to consider both sources of information. Here we consider the GCN-GRU architecture [28], where a GCN is applied to each snapshot independently to model the structural behaviour of nodes, and a Gated Recurrent Unit (GRU) [6] is then applied to the sequence of resulting embeddings to model their temporal evolution. Formally, given a subsequence of the dynamic graph $\{\mathcal{G}^t\}_{t=t^*}^T$ of length $T - t^* + 1$ starting from snapshot t^*, we obtain the embedding matrices $Z^t = \text{GCN}(\mathcal{G}^t)$ and $H^T = \text{GRU}\left(\{Z^t\}_{t=t^*}^T\right)$, whose rows are the node embeddings. We call *memory* the number of previous time steps used to build the GRU, i.e. *mem*$= T - t^*$.

Problem definition: We formulate the node and edge AD problem as a binary classification task, where, for each node u or edge ϵ, two learnable scoring functions, $f_{\text{node}}(\cdot)$ and $f_{\text{edge}}(\cdot)$, estimate their anomaly score at time t (i.e. the probability of class *anomalous*) and the true labels y_ϵ^t, y_u^t are either given or generated via self-supervision. The edge anomaly score depends on a subset of the dynamic graph, i.e. $s_\epsilon^t = f_{\text{edge}}(\epsilon, \{\mathcal{G}^{t'}\}_{t'=t-mem-1}^t) \in [0, 1]$, where a higher value indicates a higher level of anomaly. Analogously, $s_u^t = f_{\text{node}}(u, \{\mathcal{G}^{t'}\}_{t'=t-mem-1}^t) \in [0, 1]$ for node anomaly score.

3 Related Works

GNNs for AD: GNNs have been widely used for AD on graphs [23]. Several methods model the normal behaviour of a network and classify as anomalous the entities that deviate from standard patterns, targeting either anomalous nodes or anomalous edges. Node AD is mainly performed on static graphs [7, 21], whereas edge AD is also analysed on dynamic networks [22, 23]. However, existing works evaluate their models only on synthetically injected anomalies, that may not reflect real-world anomalies in application scenarios. In this work, instead, we explicitly focus on node AD for dynamic graphs and we consider also real anomalies.

AD in communications networks: AD in communications networks is widely studied [1, 15, 25, 27]. Previous works covered a wide range of methods, from statistical and knowledge-based rules to traditional machine learning methods such as clustering, SVM or random forest. Recently, deep learning approaches were proposed and GNNs proved to be an effective tool for several tasks on communications networks which can be framed as an AD problem, such as intrusion detection [2, 13]. Most works focus on edge AD, i.e. identifying anomalous connections only [5, 16]. The problem of node AD on dynamic graphs is less studied, despite its relevant applications – e.g. differently from our approach, some works perform host intrusion detection [19, 20] focusing on host-level data such as system logs using static GNNs. Here, we use tGNNs leveraging the natural evolution of the networks.

4 Methodology

Data preprocessing: We consider lists of interactions with their time of occurrence (e.g. the list of packets or flows exchanged between two nodes, the list of interactions among users in a social network, the list of API requests from web applications). We split them into independent snapshots, each collecting the interactions happened in a given time interval, which we represent as a weighted undirected graph $\mathcal{G}^t$, where edge weights are the number of times the event occurred during the snapshot t.

Anomaly scores: We generate an embedding h_u^t for each node u at snapshot t using GCN-GRU given its simplicity and effectiveness (note that our pipeline can incorporate different tGNNs). Then, we generate anomaly scores. For an edge $\epsilon = (u, v, w)$, we apply a learnable scoring function $\phi_{\text{edge}}(\cdot)$ to the embeddings at the previous snapshot of the two incident nodes of the edge, i.e. $s_\epsilon^t = s_{(u,v)}^t = \phi_{\text{edge}}(h_u^{t-1}, h_v^{t-1})$. Note that we do not compute the node embeddings at snapshot t because they would be affected by the anomalous connections. We assume that anomalous nodes perform a relevant number of unexpected connections; thus, their neighbourhood differs from standard patterns [21]. Therefore, for each node u at snapshot t, we compute its neighbourhood embedding n_u^t as a weighted sum of the neighbours' embeddings, i.e. $n_u^t = \sum_{v \in \mathcal{N}(u)^t} a_{uv}^t h_v^{t-1} / |\mathcal{N}(u)^t|$. The coefficients $a_{uv}^t \in (0, 1)$ are obtained through a learnable attention function:

$a^t_{uv} = \phi_{\text{att}}(h^{t-1}_u, h^{t-1}_v)$. This allows the network to give more importance to the most informative neighbours (e.g. the anomalies). In conclusion, we obtain the anomaly score through a learnable scoring function $\phi_{\text{node}}(\cdot)$ that receives as input the node and neighbourhood embeddings: $s^t_u = \phi_{\text{node}}(h^{t-1}_u, n^t_u)$. To implement the learnable functions $\phi_{\text{edge}}, \phi_{\text{node}}$ (referred to as AD-NNs) and ϕ_{att} we employ Multi-Layer Perceptrons (MLPs).

Model training: To remove the need for labelled data, we use self-supervised training. For edge AD, at each training snapshot t, we compute node embeddings H^{t-1} and use them for *link prediction*. Real edges $\mathcal{E}^t$ are positive examples, i.e. normal ($y^t_\epsilon = 0 \ \forall \epsilon \in \mathcal{E}^t$). We sample an equal number of non-existing edges $\overline{\mathcal{E}^t}$ as negative examples ($y^t_\epsilon = 1 \ \forall \epsilon \in \overline{\mathcal{E}^t}$), i.e. anomalies. As training loss, we employ the negative log-likelihood: $\mathcal{L}^t_{\text{edges}} = -\sum_{\epsilon \in \mathcal{E}^t \cup \overline{\mathcal{E}^t}} (y^t_\epsilon \log(s^t_\epsilon) + (1 - y^t_\epsilon) \log(1 - s^t_\epsilon))$, thus training the NN to assign scores s^t_ϵ close to 0 (1) for normal (anomalous) edges. For node AD, at each training snapshot t, and given node embeddings H^{t-1}, we generate for each node $u \in \mathcal{V}^t$ one positive neighbourhood embedding n^t_u with the real neighbours and one negative neighbourhood embedding $\overline{n^t_u}$ with random non-neighbouring nodes. Therefore, for each node u, we obtain two anomaly scores: $s^t_u = \phi_{\text{node}}(h^{t-1}_u, n^t_u)$ and $\overline{s^t_u} = \phi_{\text{node}}(h^{t-1}_u, \overline{n^t_u})$, as reported in Figure 1. As loss, we employ the negative log-likelihood: $\mathcal{L}^t_{\text{nodes}} = -\sum_{u \in \mathcal{V}^t} (\log(\overline{s^t_u}) + \log(1 - s^t_u))$. Again, the network learns to assign a low (high) anomaly score s^t_u ($\overline{s^t_u}$) when u's neighbourhood embedding is real (fake). We train the overall NN (tGNN+AD-NN) end-to-end with three configurations: (i) *edge-only* AD: minimise only $\mathcal{L}^t_{\text{edges}}$; (ii) *node-only* AD: minimise only $\mathcal{L}^t_{\text{nodes}}$; (iii) *multitask (MT)* AD: minimise $\mathcal{L}^t_{\text{MT}} = \lambda \cdot \mathcal{L}^t_{\text{nodes}} + (1 - \lambda) \cdot \mathcal{L}^t_{\text{edges}}$, where λ is a hyperparameter. We refer to *edge-only* and *node-only* as *specialised models*. We optimize the weights of the GNN, GRU and MLPs over a sequence of training graphs using back-propagation with Adam optimizer for 10 epochs.

Hyperparameters tuning: We split the available temporal snapshots into four sets: initial warm-up snapshots $t \in [1, T_{warm}]$, training $t \in [T_{warm} + 1, T_{train}]$, validation $t \in [T_{train} + 1, T_{val}]$ and test $t \in [T_{val} + 1, T_{test}]$. The initial warm-up set provides the model with historical information at the first training snapshot. After training the NN, we use the validation snapshots to find the best set of hyperparameters. Then, we evaluate the model on the test set.

5 Datasets and Anomalies

5.1 Synthetic datasets with synthetic anomalies

We generate two synthetic datasets to analyse how our models perform when the graph structure and evolution are controlled.

Erdős-Rényi. Based on the Erdős-Rényi model [8], we generate a random graph in which nodes are connected with a fixed probability. Each edge has a predefined uniformly random integer weight within the range $[45, 55]$. We generate this initial graph with 10 000 nodes and 50 000 edges. We then generate 40 sequential snapshots by perturbing the initial graph through (i) *weights perturbation*, i.e. we add an integer within $[-5, 5]$ to the weights of 30% of the edges; (ii) *edge shuffling*, i.e. we randomly choose 30% of existing edges and reassign one of the incident nodes at random; and (iii) *on-off nodes*, i.e. we remove each node (and all its edges) for one snapshot with 30% probability. We perform anomaly injection only on validation and test snapshots by transforming 5% of the active nodes into anomalies. We add to each of them between 2 and 6 new edges to random destinations with a weight uniformly distributed in the range $[60, 70]$. The added edges become edge anomalies.

Bipartite. Inspired by Client-Server communications, we generate a bipartite graph with a Server layer (50 groups of 5 nodes) and a Client layer (50 groups of a uniformly distributed random number of nodes within $[200, 250]$). Each group of servers serves the requests of one group of clients, whose members are connected to all corresponding 5 servers, with a random weight within $[50, 70]$. This pattern also mimics social media networks where influencers (servers) and followers (clients) interact. Over 40 snapshots, we apply (i) *weights perturbation* and (ii) *on-off nodes* as in Erdős-Rényi. On validation and test snapshots, we select 5% of the active clients and add them between 2 and 6 new anomalous edges towards servers in different groups to make them anomalous.

5.2 Real datasets with injected anomalies

We focus on 4 public real-world datasets. These are time-evolving graphs for which we inject synthetic anomalies for validation and test. ***Reddit*** [18] is a bipartite graph in which nodes represent users (user layer) that posted on the top 1 000 subreddits and edges represent a post of a user on a specific subreddit. On average, the graph has about 7 000 nodes and 20 000 edges per snapshot. **WebBrowsing (WB)** [11] is the collection of Internet browsing histories of ≈600 users obtained through the EasyPIMS tool [12]. We create a bipartite graph where nodes are website users and edges indicate a user's visit to a website. On average, the graph has about 2 500 nodes and 8 000 edges per snapshot. ***StackOverflow (SO) [24]*** is a graph in which nodes are StackOverflow users and edges are interactions (answers or comments). On average, the graph has about 7 000 nodes and 10 000 edges per snapshot. ***UCI** [22]* collects messages exchanged on an online social network between students at the University of California, Irvine. Nodes represent users and edges represent the exchanged messages. On average, the graph has about 300 nodes and 2 000 edges per snapshot. All datasets except UCI contain one month of data and we create snapshots of 1 day. UCI, instead, contains ≈4 months of data and we generate snapshots of 4 days. In this way, we generate 31 snapshots for each dataset.

We inject the following anomalies for validation and test.

Node anomalies: At snapshot t, we add to 5% of the active nodes $\max(4, \text{ Uniform}[\ |\mathcal{N}(u)^t|/2, \ |\mathcal{N}(u)^t|\ ])$ random anomalous connections, such that node anomalies perform a sizeable number of anomalous connections but are not straightforward to identify. On bipartite graphs, we select anomalies in the user layer.

Edge anomalies: At snapshot t, we further add $0.05 \cdot |\mathcal{E}^t|$ random edges. We inject edge anomalies unrelated to node anomalies to test whether our node AD strategy signals nodes only if they perform significant anomalous activity.

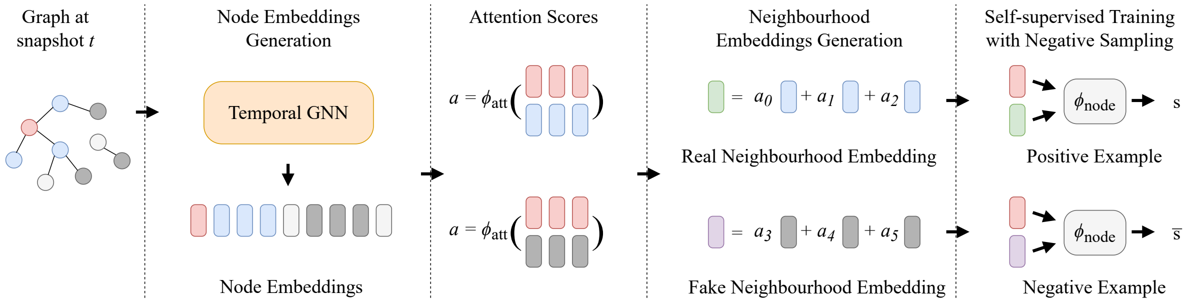

Figure 1: Node anomaly score computation at snapshot t. Given a target node (red), we compute the green weighted sum of the embeddings of its neighbours (in blue). Similarly, we compute the violet weighted sum of the embeddings of random nodes (non-neighbours – in grey) for self-supervision. Finally, the classifier ϕ_{node} computes the scores given the node and its neighbourhood embeddings.

5.3 Real datasets with real anomalies

***LANL** [26]* collects authentication events from desktop computers and servers at Los Alamos National Labs. In the graph, nodes represent network hosts and edges represent authentication events. We partition this graph into 501 snapshots of 10 000 seconds following [16]. The ground truth consists of authentication events performed by a red team. We use these as anomalous edges and we define as anomalous nodes the hosts with at least 4 anomalous connections in a snapshot. Here, the fraction of anomalies is very low (0.0007% for nodes and 0.008% for edges).

***Darknet** [10]* contains one month of traffic traces collected from a /24 darknet hosted at Politecnico di Torino. We build a bipartite graph where nodes represent sender hosts and destination TCP ports. Edges represent packets sent from hosts to ports. The ground truth is available for a subset of the nodes and identifies groups of known hosts belonging to online security services, research projects and known botnets that run daily massive internet scans targeting the Internet (e.g. Mirai). We only consider hosts for which the ground truth is available. Since all traffic observed by darknets is anomalous by definition, we evaluate our models' capability to detect different types of anomalous behaviour. Specifically, we remove all hosts belonging to one known class from the training snapshot and we enable them in validation and test snapshots. We repeat this process 5 times, each time with a different ground truth class. In this way, the AD performance can be interpreted as the ability to detect a new scan campaign run by a novel group of hosts.

6 Experiments

6.1 Experimental setup

We assume the training set contains clean data (i.e. no anomalies). This might not be the case for the real datasets, and in Section 6.4 we analyze the impact of injected anomalies also in training data. For LANL, we use the first 5 snapshots as initial warm-up snapshots, snapshots 6-14 for training (since they do not contain anomalous events), snapshots 15-86 for validation and the remaining for test.[1] For all the other datasets, we use the first 11 snapshots as initial warm-up, snapshots 12-21 for training, snapshots 22-26 for validation and the remaining for testing. Since we do not consider node features, we set $e_f = |\bigcup_t \mathcal{V}^t|$ and feature matrix $X = I$, with $I \in \mathbb{R}^{e_f \times e_f}$, as common in similar settings [4, 29]. On each dataset, we optimize the hyperparameters for the *MT* model and use the same configuration for the specialised ones. In particular, we use a GCN with a single hidden layer ($L = 1$) with size within $\{2048, 1024, 512, 128\}$ and output size $e^{in} \in \{1024, 512, 128\}$. We use a single-layer GRU with output size $e^{out} \in \{128, 64, 32\}$ (final embedding size). The GCN-GRU considers the past *mem*+1 snapshots, where $mem \in [0, 10]$. The parameter λ is within $\{0.1, 0.3, 0.5, 0.7, 0.9\}$. The MLPs that produce the scores have a single hidden layer with size within $\{64, 32\}$. The choice of the hyperparameters is often not critical, i.e. different configurations result in limited differences in performance. We run experiments on a Tesla V100-PCIE-16GB.

6.2 Baselines

Global Rule Based (GRB): rule-based method that considers node degree distribution over the complete graph (global). For node AD, we create an embedding for each node by concatenating its degree in the past snapshots. For edges, we create an embedding by concatenating the degrees of the two nodes they connect. We then use the Local Outlier Factor algorithm [3] to detect anomalies.

Local Rule Based (LRB): analogous to GRB (rule-based), but we normalize degrees by the neighbours' cumulative degree (local).

Multitask without memory - MT(mem=0): static GNN that considers only information from the previous snapshot (i.e. $mem = 0$).

Heuristics: use a specialised model for the opposite task. The *edge heuristic* computes the anomaly score for a node u as the average of the scores of its incident edges computed by the *edge-only* model, i.e. $s_u = \sum_{v \in \mathcal{N}(u)} s_{(u,v)} / |\mathcal{N}(u)|$. The *node heuristic* computes the anomaly score for an edge $\epsilon = (u, v, w)$ as the average of the scores computed by the *node-only* model for the two nodes the edge connects, i.e. $s_{(u,v)} = (s_u + s_v)/2$.

6.3 Main results

Table 1 compares the AUC of different models on node and edge AD. We report the average ranking (between 1st and 6th) of each model over datasets and tasks. The rule-based detectors (GRB and LRB) generally perform very poorly, thus highlighting the need for more elaborate techniques that detect more subtle anomalies. Similarly,

[1] We split validation and test sets such that they have a similar number of anomalies.

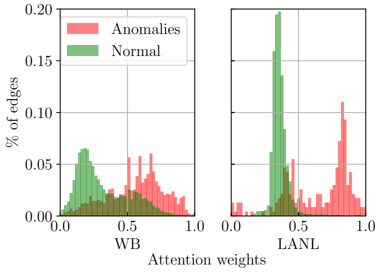

Figure 2: Distribution of the attention weights learned by the *MT* model on WebBrowsing (WB) and LANL.

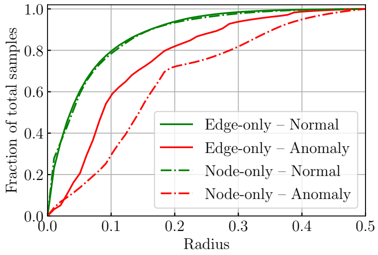

Figure 3: Fraction of samples within a radius for normal and anomalous nodes. Results for a WB test snapshot.

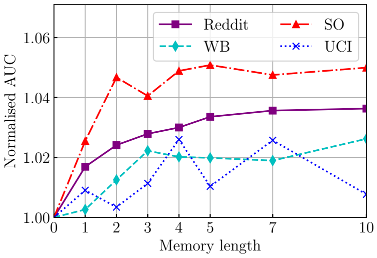

Figure 4: Normalised AUC for edge AD on 4 datasets with different values of *mem*. Results for the *MT* model.

Table 1: Average AUC and standard deviation over 5 runs for different models on edge and node AD on different datasets. Best results for each task are in bold, results within the standard deviation interval of the best are in blue.

		Edge Anomaly Detection					
		LRB	GRB	MT(*mem*=0)	Node heur.	Edge-only	MT(*mem**)
Syn.	Erdős-Rényi	0.446	0.445	0.827±0.006	0.797±0.082	**0.892±0.002**	0.863±0.007
	Bipartite	0.719	0.600	0.892±0.004	0.896±0.035	**0.992±0.002**	0.989±0.002
Injected	Reddit	0.500	0.491	0.908±0.002	0.794±0.024	**0.942±0.002**	0.941±0.002
	WebBrowsing	0.530	0.462	0.822±0.007	0.552±0.107	**0.845±0.006**	0.840±0.002
	StackOverflow	0.467	0.475	0.618±0.010	0.553±0.012	0.644±0.003	**0.649±0.004**
	UCI	0.332	0.496	0.770±0.015	0.700±0.014	**0.800±0.012**	0.790±0.016
Real	LANL	0.480	0.616	**0.960±0.004**	0.508±0.299	0.948±0.012	0.958±0.002
	Darknet[†]	0.481	0.489	**0.833±0.011**	0.694±0.077	0.821±0.026	0.826±0.032
	Ranking	5.5	5.4	2.7	4.0	1.6	1.9

		Node Anomaly Detection					
		LRB	GRB	MT(*mem*=0)	Node-only	Edge heur.	MT(*mem**)
Syn.	Erdős-Rényi	0.520	0.594	0.747±0.006	0.759±0.006	0.632±0.003	**0.769±0.005**
	Bipartite	0.496	0.745	0.846±0.019	0.903±0.029	**0.984±0.001**	0.976±0.006
Injected	Reddit	0.504	0.353	0.873±0.004	0.896±0.004	**0.912±0.002**	0.898±0.002
	WebBrowsing	0.746	0.734	0.963±0.009	**0.973±0.002**	0.477±0.051	0.972±0.005
	StackOverflow	0.463	0.656	0.654±0.008	0.656±0.009	0.570±0.003	**0.671±0.010**
	UCI	0.502	0.738	0.799±0.007	0.807±0.013	0.738±0.009	**0.814±0.020**
Real	LANL	0.752	0.989	0.979±0.012	0.984±0.004	0.984±0.006	**0.992±0.001**
	Darknet[†]	0.421	0.421	**0.855±0.024**	0.830±0.024	0.819±0.025	0.819±0.028
	Ranking	5.6	4.6	3.2	2.2	3.5	1.8
	Overall ranking	5.6	5.0	3.0	3.1	2.6	**1.9**

[†] Average over the 5 experiments using different ground truth classes as anomalies.

the performance of the heuristics is generally inconsistent. In some cases, they achieve the best results or are comparable with the best model (e.g. node AD on Reddit and Bipartite), but, in others, they perform very poorly (e.g. edge AD on WebBrowsing, StackOverflow and LANL, node AD on WebBrowsing). This justifies the need for a more sophisticated model to detect anomalous nodes and edges or both, depending on the application. Static models (*MT(mem=0)*) generally achieve worse results than dynamic models, especially on edge AD (up to ≈0.10 AUC difference). This demonstrates the impact of considering past information to model the evolution of the evolving networks over time. Darknet is an exception, as static models perform better than temporal ones, which might be due to the highly dynamic behaviour of nodes [10] that makes historical information not reliable. The *MT* model tends to perform comparably to the specialised models and, overall, is the best-ranked model across tasks. On node AD on Bipartite, the *MT* model significantly improves over the *node-only* model. On this graph, edge detectors perform very well, whereas node-based methods struggle to even perform comparably to the *edge heuristic*. Here, the knowledge introduced by an edge-based loss in multitask training provides benefits also to the node AD task. StackOverflow proves to be the most complicated scenario for AD. This shows that this dataset is less structured and more heterogeneous than the others, and therefore the injected anomalies are less evident. In conclusion, the solid results on LANL and Darknet demonstrate that the models are effective also with real-world anomalies. Specifically, all methods except the *node heuristic* perform very well on LANL, showing that the proposed approach models intrusion detection successfully. On Darknet, we are able to identify the sudden arrival of new patterns over repeated experiments.

6.4 Additional results and sensitivity

Attention scores: Figure 2 shows the distribution of the attention weights a^t_{uv} computed by the *multitask* model. Anomalous edges receive higher attention than normal connections, thus contributing more to the neighbourhood embeddings. In a nutshell, the model exploits the attention mechanism to assign higher importance to edges that carry more information for the AD task. Moreover, the attention weights provide interpretability as they let us understand what the model focuses on when making decisions. This is fundamental for a system to be successfully applied in practice, as practitioners can observe which of an anomalous host's interactions were more suspicious.

Evaluation of the embeddings: We further analyse the node embeddings generated by the two specialised models (*edge-only* and *node-only*) on one test snapshot of WebBrowsing. For each node embedding in the latent space, we count the number of samples whose cosine distance is lower than a given radius. The lower the number, the more isolated the observation is. Figure 3 shows that anomalous nodes are more isolated in the latent space than normal ones, since their percentage of close samples grows more slowly with increasing radius. This is more evident for the *node-only* model, as the training loss $\mathcal{L}^t_{\text{nodes}}$ directly works on the scores of the nodes.

Table 2: Average AUC and difference with respect to the main results in Table 1 (red) for experiments with anomalies injected also in the training set.

	Edge Anomaly Detection		Node Anomaly Detection	
	Edge-only	MT	Node-only	MT
Reddit	0.918(-0.024)	0.925(-0.016)	0.867(-0.029)	0.871(-0.027)
WB	0.825(-0.020)	0.821(-0.019)	0.967(-0.006)	0.955(-0.017)

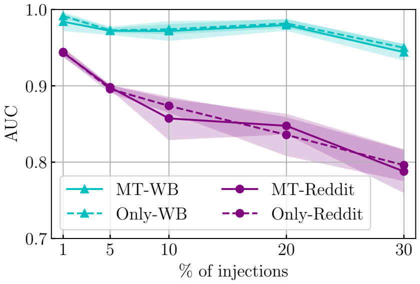

Figure 5: Average AUC and std with different percentages of injected anomalies for node AD.

This is expected since normal nodes are more numerous and their behaviour is more consistent.

Presence of anomalies during training: To evaluate the applicability of our method to real-world cases where it might be impossible to train on clean data, we assess the detection ability when the anomalies are present *also* during training. The injection procedure in the training snapshots is the same used for the validation and test snapshots. The results in Table 2 show that the performance only marginally decreases compared to the main results. This demonstrates the robustness of our methods to the presence of anomalies in the training set. Note that the anomalies injected in the training set are not correlated to the test set ones, as we assume that, in a real setting, one might train the model on potentially anomalous past data to identify new different anomalies in the future.

Impact of memory: We investigate the importance of the temporal component of tGNNs in Figure 4. In general, increasing the memory length improves the results until reaching a saturation point. This highlights the importance of considering historical information through time-aware GNNs which model the evolution of the network over time. UCI shows a less evident gain as the network evolves more quickly than others.

Sensitivity against volume of anomalies: Figure 5 shows how different quantities of injected anomalies affect the performance. As expected, a larger percentage of injected anomalies leads to slightly lower AUC, as it is harder for the model to identify the normal behaviour of nodes in the network. This happens for both the *MT* and the specialised *node-only* models.

7 Conclusions

In this work we presented an analysis of temporal GNN-based approaches for node and edge AD on communications networks. Through extensive experiments, we demonstrated the effectiveness of these methods on synthetic and real-world datasets with artificial and real anomalies. This analysis represents a first step towards the effective implementation of GNN-based anomaly detectors in practical applications. In fact, (i) we explicitly faced the natural and highly-dynamic temporal evolution of the communication processes thanks to the memory ability of tGNNs; (ii) we used a completely self-supervised learning approach that requires no ground truth labels; (iii) we focused on both edge and node anomalies, showing the advantages of facing both tasks simultaneously.

Future developments of the work include the analysis of different temporal GNN architectures, anomaly scoring functions and training strategies. Moreover, our pipeline can be easily extended to include node features as additional network information. This could lead to the definition of other types of anomalies that depend not only on the node activity within the network but also on their characteristics (e.g. servers rather than clients). Similarly, we could extend the anomaly scoring process to account for edge weights and, consequently, define new types of anomalies based on the values of the connections.

Acknowledgments

This work was supported by the project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union (NextGenerationEU).

References

[1] Mohiuddin Ahmed, Abdun Naser Mahmood, and Jiankun Hu. 2016. A survey of network anomaly detection techniques. *Journal of Network and Computer Applications* 60 (2016), 19–31.

[2] Tristan Bilot, Nour El Madhoun, Khaldoun Al Agha, and Anis Zouaoui. 2023. Graph Neural Networks for Intrusion Detection: A Survey. *IEEE Access* 11 (2023), 49114–49139.

[3] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. 2000. LOF: identifying density-based local outliers. *SIGMOD Rec.* 29, 2 (may 2000), 93–104.

[4] Andrea Cavallo, Claas Grohnfeldt, Michele Russo, Giulio Lovisotto, and Luca Vassio. 2022. 2-hop Neighbor Class Similarity (2NCS): A graph structural metric indicative of graph neural network performance. *arXiv preprint arXiv:2212.13202* (2022).

[5] Evan Caville, Wai Weng Lo, Siamak Layeghy, and Marius Portmann. 2022. Anomal-E: A self-supervised network intrusion detection system based on graph neural networks. *Knowledge-Based Systems* 258 (2022), 110030.

[6] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 1724–1734.

[7] Kaize Ding, Jundong Li, Rohit Bhanushali, and Huan Liu. 2019. Deep Anomaly Detection on Attributed Networks. In *SIAM International Conference on Data Mining (SDM)*.

[8] P. Erdős and A. Rényi. 1959. On Random Graphs I. *Publicationes Mathematicae Debrecen* 6 (1959), 290.

[9] Luca Gioacchini, Andrea Cavallo, Marco Mellia, and Luca Vassio. 2023. Exploring Temporal GNN Embeddings for Darknet Traffic Analysis. In *Proceedings of the 2nd on Graph Neural Networking Workshop 2023* (Paris, France) *(GNNet '23)*. Association for Computing Machinery, New York, NY, USA, 31–36. https://doi.org/10.1145/3630049.3630175

[10] Luca Gioacchini, Luca Vassio, Marco Mellia, Idilio Drago, and Zied Ben Houidi. 2023. i-DarkVec: Incremental Embeddings for Darknet Traffic Analysis. *ACM Trans. Internet Technol.* (2023).

[11] Nikhil Jha, Martino Trevisan, Emilio Leonardi, and Marco Mellia. 2023. On the Robustness of Topics API to a Re-Identification Attack. *arXiv preprint*

arXiv:2306.05094 (2023).

[12] Nikhil Jha, Martino Trevisan, Luca Vassio, Marco Mellia, Stefano Traverso, Alvaro Garcia-Recuero, Nikolaos Laoutaris, Amir Mehrjoo, Santiago Andrés Azcoitia, Ruben Cuevas Rumin, Kleomenis Katevas, Panagiotis Papadopoulos, Nicolas Kourtellis, Roberto Gonzalez, Xavi Olivares, and George-Marios Kalatzantonakis-Jullien. 2022. A PIMS Development Kit for New Personal Data Platforms. *IEEE Internet Computing* 26 (2022), 79–84.

[13] Weiwei Jiang. 2022. Graph-based deep learning for communication networks: A survey. *Computer Communications* 185 (2022), 40–54.

[14] Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. 2020. Representation Learning for Dynamic Graphs: A Survey. *Journal of Machine Learning Research* 21, Article 70 (2020), 73 pages.

[15] Ansam Khraisat, Iqbal Gondal, Peter Vamplew, and Joarder Kamruzzaman. 2019. Survey of intrusion detection systems: techniques, datasets and challenges. *Cybersecurity* 2 (2019), 1–22.

[16] Isaiah J. King and H. Howie Huang. 2023. Euler: Detecting Network Lateral Movement via Scalable Temporal Link Prediction. *ACM Transactions on Privacy and Security* 26, Article 35 (2023), 36 pages.

[17] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations (ICLR)*.

[18] Srijan Kumar, Xikun Zhang, and Jure Leskovec. 2019. Predicting Dynamic Embedding Trajectory in Temporal Interaction Networks. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*.

[19] Zitong Li, Xiang Cheng, Lixiao Sun, et al. 2021. A Hierarchical Approach for Advanced Persistent Threat Detection with Attention-Based Graph Neural Networks. *Security and Communication Networks* 2021 (2021), 14 pages.

[20] Fucheng Liu, Xihe Jiang, Yu Wen, Xinyu Xing, Dongxue Zhang, and Dan Meng. 2019. Log2vec: A heterogeneous graph embedding based approach for detecting cyber threats within enterprise. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*.

[21] Yixin Liu, Zhao Li, Shirui Pan, Chen Gong, Chuan Zhou, and George Karypis. 2021. Anomaly detection on attributed networks via contrastive self-supervised learning. *IEEE transactions on neural networks and learning systems* 33 (2021), 2378–2392.

[22] Yixin Liu, Shirui Pan, Yu Guang Wang, Fei Xiong, Liang Wang, Qingfeng Chen, and Vincent CS Lee. 2021. Anomaly detection in dynamic graphs via transformer. *IEEE Transactions on Knowledge and Data Engineering* (2021).

[23] Xiaoxiao Ma, Jia Wu, Shan Xue, Jian Yang, Chuan Zhou, Quan Z Sheng, Hui Xiong, and Leman Akoglu. 2021. A comprehensive survey on graph anomaly detection with deep learning. *IEEE Transactions on Knowledge and Data Engineering* (2021).

[24] Ashwin Paranjape, Austin R. Benson, and Jure Leskovec. 2017. Motifs in Temporal Networks. In *ACM International Conference on Web Search and Data Mining* (Cambridge, United Kingdom).

[25] Francesca Soro, Thomas Favale, Danilo Giordano, Luca Vassio, Zied Ben Houidi, and Idilio Drago. 2021. The New Abnormal: Network Anomalies in the AI Era. *Communication Networks and Service Management in the Era of Artificial Intelligence and Machine Learning* (2021), 261–288.

[26] Melissa J. M. Turcotte, Alexander D. Kent, and Curtis Hash. 2018. *Unified Host and Network Data Set.* World Scientific, 1–22.

[27] Rose Yu, Huida Qiu, Zhen Wen, ChingYung Lin, and Yan Liu. 2016. A survey on social media anomaly detection. *ACM SIGKDD Explorations Newsletter* 18 (2016), 1–14.

[28] Ling Zhao, Yujiao Song, Chao Zhang, Yu Liu, Pu Wang, Tao Lin, Min Deng, and Haifeng Li. 2020. T-GCN: A Temporal Graph Convolutional Network for Traffic Prediction. *IEEE Transactions on Intelligent Transportation Systems* 21 (2020), 3848–3858.

[29] Jiong Zhu, Ryan A. Rossi, Anup Rao, Tung Mai, Nedim Lipka, Nesreen K. Ahmed, and Danai Koutra. 2021. Graph Neural Networks with Heterophily. *AAAI Conference on Artificial Intelligence* (2021).