Detecting Abnormal Communication Patterns in IoT Networks Using Graph Neural Networks

Vincenzo Carletti(✉), Pasquale Foggia, and Mario Vento

Department of Information Engineering, Electrical Engineering and Applied Mathematics, University of Salerno, Fisciano, Italy
{vcarletti,pfoggia,mvento}@unisa.it
https://mivia.unisa.it

Abstract. Nowadays, millions of Internet of Things (IoT) devices communicate over the Internet, thus becoming potential targets for cyber-attacks. Due to the limited hardware capabilities of these devices, host-based countermeasures are unlikely to be deployed on them, making network traffic analysis the only reasonable way to detect malicious activities. In this paper, we face the problem of identifying abnormal communications in IoT networks using graph-based anomaly detection methods. Although anomaly detection has already been applied to graph-based data, most existing methods have been used for static graphs, with the aim of detecting anomalous nodes. In our case, the graphs represent snapshots of the network traffic, and change with time. In this paper we compare different graph-based methods, and different graph representations of the network traffic, using two large datasets of real IoT data.

Keywords: Network Anomaly Detection · IoT Networks · Graph Neural Networks

1 Introduction

The Internet of Things (IoT) has revolutionized different fields leading the development of smart homes, smart medical devices, smart cities and smart industries. IoT devices are embedded with sensors and communication technologies that enable them to collect and transmit data over the internet. However, the wide diffusion of such devices, together with their limited security capabilities, has made them the preferred target of malicious users for performing Distributed Denial of Service attacks (DDoS) [15]. Common host-based countermeasures, like intrusion detection systems (HIDS) or antivirus software are unlikely to be installed on IoT devices, due to their limited hardware capabilities. Therefore, the only reasonable way to detect infected devices or malicious activities is to analyze the network exchanges observed by a dedicated monitoring device. Unfortunately, IoT network communications are highly dynamic and heterogeneous, if compared with traditional personal computer networks, making more challenging for traditional network-based intrusion detection systems (NIDS) to detect malicious traffic.

M. Vento et al. (Eds.): GbRPR 2023, LNCS 14121, pp. 127–138, 2023.
https://doi.org/10.1007/978-3-031-42795-4_12

Several methods for network analysis, based on machine learning and deep learning, have been proposed [1,10,12,13], but most of them assume that the characteristics of malicious traffic are known in advance (or, at least, can be inferred from a set of training examples). Of course, this assumption does not hold for many real-world scenarios, where we are not aware in advance of all the possible threats, and it is inpractical to collect a sufficient amount of samples of malicious traffic.

An alternative is to deal with this problem as an anomaly detection task, where the system learns a model of normal traffic, so as to classify anything that does not fit this model as a threat [6]. Anomaly detection methods for IoT networks have been reviewed and compared in some recent papers [3,6,8]. Most of them work with vectors of measurements extracted from *network flows*, i.e. sequences of network packets sharing the same transport protocol and endpoints. Although some network flows statistics have been proved to be very effective to detect anomalies in the communication between two network nodes, they seem to be less discriminant for abnormal communication patterns involving larger groups of devices [9], as in the case of botnets.

Graph-based approaches can help to overcome this limitation: a graph is a natural representation for data where the *structure*, i.e. the way the different pieces are interconnected, plays an important role.

In the scientific literature, different Graph Neural Networks (GNNs) have been proposed to detect anomalous nodes in attributed graphs [11,18]. Among these, we have selected three different approaches recently proposed, for which the authors publicly provide an implementation. DOMINANT (Deep Anomaly Detection on Attributed Networks) [5], a deep graph autoencoder where the encoder function aims to map node features to a lower-dimensional latent space, and the decoder function aims to reconstruct both the original feature matrix and the graph topology using these compressed node representations. CONAD (Contrastive Anomaly Detection) proposed in [18] to identify anomalous nodes in attributed graphs by utilizing prior knowledge. Together with the GNN the authors have also proposed a new approach to graph data augmentation that explicitly incorporates human knowledge of various anomaly types as contrastive samples. The latter correspond to nodes whose structural and semantic information deviate considerably from those of existing nodes in the input attributed network. The contrastive samples are used to generate an adversarial version of the input graph. OCGNN (One Class Graph Neural Network), a graph version of the One Class Support Vector Machine OCSVM, proposed by Wang et al. in [17]. The method aims at learning the minimum volume hyper-sphere that contains all the embeddings of normal nodes.

These graph-based methods face the detection of abnormal communication patterns as a node classification problem and are trained in a transductive context, that involves training and testing the network on a single large attributed graph. Although this is a very common approach, it does not allow to take into account the dynamic nature of the IoT networks over time. To face this limitation, in this paper, we have partitioned the network communication data into temporal snapshots as proposed in [21]. Each snapshot is represented as a distinct graph.

For the graph representation of a snapshot, we have considered three approaches: *similarity graphs* [4], that provide a general method to transform time series into graphs; *traffic trajectory graphs* [19,20], a graph representation specifically devised for network flows; and *extended TDG* [21], an extension of Traffic Dispersion Graphs [7] (TDGs) originally presented to model the behavior of network hosts. The overall network traffic is thus represented through a sequence of graphs, having a topology that can vary according to the communication flows in the corresponding temporal snapshot.

For each of the three graph-based representations, we have adapted the three mentioned GNN anomaly detection methods, modifying their learning approach from transductive (i.e. the GNN learns from different subsets of the same graph) to inductive (i.e. each training sample is a different, complete graph). The 9 resulting combinations of graph representation and GNN method have been experimentally evaluated using two large, recent datasets of real IoT traffic.

The paper is organized as follows. In Sect. 2, we describe the graph-based representations for network traffic snapshots. In Sect. 3, we provide some details on the three GNN considered. In Sect. 4, we describe the experimental setup, the used datasets and how we have trained the GNN; then we present, compare and discuss the results obtained by each combination of GNN and graph representation. Finally, in Sect. 5 we draw our conclusions about the effectiveness of using GNN for the task at hand.

2 Representing Network Traffic as a Graph

Although representing the static topology of a network as a graph is a relatively straightforward process, in which hosts are assigned to nodes and physical links to edges, there is not a unique way for representing network communications as graphs. As introduced in Sect. 1, we have selected three recent graph-based representations that are suitable for the task at hand.

Starting from the captured raw packets, we have extracted the communication flows and the related feature vectors using NFStream [2], a publicly available tool for traffic analysis. Each feature vector represents a flow, i.e. a sequence of packets having the same transport-level endpoints; the vectors contain 77 features commonly used to represent flows in network analysis, including categorical features like IP and MAC address, statistical features such as average packet size or number of packets as well as temporal features like duration and starting time.

During the extraction, in order to represent the evolution of the communication over time, as proposed in [21], we have marked each network flow with the timestamp obtained from its first captured packet and have partitioned the overall network communication in time windows of duration T; so that all the flows having the timestamp that falls in the time window $(t, t+T)$ have been grouped in the temporal snapshots $\texttt{snap}_t$.

The feature vectors of each snapshot are the input data for building the graphs, according to the methods described in the following subsections.

2.1 Similarity Graphs

Given the snapshot $\texttt{snap}_t$, a *similarity graph* [4] is made by assigning each feature vector x_i to a different node; thus nodes do not represent network devices, but communication flows.

In the next step, a node similarity matrix is obtained by computing for each pair of nodes the cosine distance between the vectors associated to the nodes. Finally, for each node, K directed edges are added connecting it to its K closest neighbors with respect to the similarity matrix.

Although similarity graphs are a general way to represent time series as graphs, and they have not been developed to model network traffic, the rationale behind their use in our application is that the flows corresponding to normal traffic should form large, densely connected clusters of nodes, while the anomalous flows will likely become nodes that are more isolated from the rest of the graph.

2.2 Traffic Trajectory Graphs

As in similarity graphs, also in *traffic trajectory graphs* [19,20] each node represents a communication flow. However, in this case, the (undirected) edges represent the fact that two flows share one of their endpoints network-level address. In this way, if there is a device that is very active in the network, the nodes corresponding to its flows will have a lot of connections. Thus, the graph structure implicitly encodes the activity level of different parts of the network in each snapshot.

2.3 Extended Traffic Dispersion Graphs

In *Traffic Dispersion Graphs* (TDGs), the nodes are associated to the transport-level endpoints that communicate over the network, i.e. the different IP address/transport port number pairs between which communication flows are exchanged, and the (undirected) edges represet the flows; notice that the flow feature vectors thus become attributes of the graph edges, instead of the graph nodes.

Extended TDGs [21] enrich the description of a node by adding, as node attributes, both some graph-related properties such as degree, centrality, betweeness, closeness, eccentricity, and the arithmetic mean of the flow feature vectors of edges adjacent to the node.

3 Graph Neural Networks for Anomaly Detection

In this Section we provide some details about the structure and loss functions of the GNNs mentioned in Sect. 1; we will also describe how they can be used to distinguish between normal and abnormal nodes.

3.1 DOMINANT

DOMINANT is a graph autoencoder for attributed graphs composed of a *attributed graph encoder*, a *feature reconstruction decoder*, and a *topology reconstruction decoder*. The encoder part uses a GNN with three layers, each of them followed by a ReLU activation function, that compute the embedding matrix Z, associating to each node a latent vector. More formally, given a weight matrix W, the adjacency matrix A and a feature matrix H (associating a feature vector to each node of the graph), each layer computes the following function:

$$f(H, A|W) = ReLU(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}HW) \tag{1}$$

where $\hat{A} = A + I$ and $\hat{D}$ is the diagonal matrix with $\hat{D}_{ii} = \sum_j \hat{A}_{ij}$. The overall computation of the encoder, given the feature matrix X corresponding to the input features of the nodes, is:

$$\begin{aligned} H^{(1)} &= f(X, A|W^{(0)}) \\ H^{(2)} &= f(H^{(1)}, A|W^{(1)}) \\ Z &= f(H^{(2)}, A|W^{(2)}) \end{aligned} \tag{2}$$

At the end of this process, the latent vectors associated to the nodes (the rows of matrix Z) do not depend only on the corresponding feature vectors, but also contain information from the k-hops neighborhood of each node.

The topology decoder aims at reconstructing the matrix A from Z, using the following computation:

$$A^* = sigmoid(ZZ^T) \tag{3}$$

while the feature decoder tries to reconstruct the feature matrix X from Z, using a structure that is similar to a single layer of the encoder:

$$X^* = f(Z, A|W^{(d)}) \tag{4}$$

.

The loss function is a linear combination of the reconstruction errors for the topology and the features, which are measured using the Frobenius norm of the differences between the input matrices and the reconstructed ones:

$$L = (1 - \alpha)||A - A^*||_F + \alpha||X - X^*||_F \tag{5}$$

Once the network has been trained, it is used as an anomaly detector by encoding and then reconstructing the input graph. The reconstruction error is used to compute an anomaly score and rank the nodes that are most anomalous in the network; thus the method does not return whether a node is anomalous or not but only the score assigned to each node. To our purpose, we added a final thresholding layer to classify normal and abnormal nodes. The best threshold has been selected by using the (Receiver Operating Characteristic) computed on the validation set.

3.2 OCGNN

OCGNN aims at learning together an embedding for the nodes of the graph, and a hyper-sphere (in the node embedding space) that contains all the normal nodes. The method does not depend on the choice of a particular embedding function; however, the authors suggest to use one or more layers organized as we have previously seen in Eq. 1.

The learnable weights of the embedding function $g(X, A|W)$ are represented by the matrix W (X and A are, as in the previous subsection, the node features and the adjacency matrix). The hyper-sphere is represented by its center c and its radius r. However, the center is computed simply as the average of the embedding vectors of the training nodes, so only r is actually learned.

The algorithm uses as its loss function:

$$L(r, W) = \frac{1}{\beta N} \sum_{v \in V_{tr}} \left[||g(X, A|W)_v - c||^2 - r^2 \right]^+ + r^2 + \frac{\lambda}{2} \sum_{i=1}^{K} ||W^k||^2 \tag{6}$$

where V_{tr} is the set of training nodes having cardinality N, the notation $g(\cdot)_v$ represents the row corresponding to node v of the embedding matrix computed by $g(\cdot)$, and $[\cdot]^+$ denotes a maximum operation between zero and its argument. Finally, λ is a regularization hyper-parameter while $\beta \in]0, 1[$ is a hyper-parameter used to balance the trade-off between enclosing all the embeddings in the hyper-sphere and getting the smallest radius: with the given loss function, the algorithm will try to enlarge the hyper-sphere if it contains less than a fraction β of the training samples, and to reduce it otherwise.

OCGNN uses the distance between a node embedding and the center of the hyper-sphere as a metric to provide an anomaly score. A node v is considered as anomalous if and only if $||g(X, A|W)_v - c||^2 \geq r^2$.

3.3 CONAD

CONAD is a graph autoencoder that uses data augmentation and a Siamese architecture to learn an optimal latent encoding. More specifically, given a training graph G (containing only normal nodes), CONAD generates an augmented graph G_{ano} containing artificially generated anomalous nodes. While the method does not depend on how these nodes are generated, the authors consider the following strategies: *(a)* adding a large number of random edges to a node, yielding a degree significantly higher than the average; *(b)* removing most of the connections of a node, making it mostly isolated from other nodes; *(c)* randomly modifying node features so as to be very dissimilar from their immediate neighbors; *(d)* randomly modifying node features so as to be significantly larger or significantly smaller that the values in most other nodes.

The encoder of the network is a Graph Attention Network (GAT) trained using a Siamese configuration, in which two instances of the encoder are applied on both G and G_{ano}; a contrastive loss function is used to make the corresponding embeddings obtained on the two graphs more similar to each other for the normal nodes, and more dissimilar if the node in G_{ano} was an artificially generated anomalous node:

$$L_{sc} = \frac{1}{N} \sum_{v \in V_{tr}} I_{y_v=0} \cdot ||z_v - \hat{z}_v|| + I_{y_v=1} \cdot [m - ||z_v, \hat{z}_v||]^+ \tag{7}$$

where z_v and $\hat{z}_v$ are the encodings computed for node v in G and in G_{ano} respectively, and $I_{y_v=1}$ and $I_{y_v=0}$ are the indicator functions representing the fact that node v in G_{ano} is normal or it has been modified to become anomalous, respectively. The value m is a hyper-parameter denoting the desired *margin* between the distances for normal nodes and the distances between a normal and an anomalous node.

Similarly to DOMINANT, the CONAD network has two decoders to reconstruct both the adjacency matrix A and the feature matrix X from the encoding matrix Z. Thus, the complete loss function used for training the network is a linear combination of the contrastive loss function presented above and the two reconstruction errors. Finally, once the network has been trained, the latter is used to decide if a node is anomalous, in a way similar to DOMINANT.

4 Experiments

In this section we describe in details the experiments conducted and the obtained results.

4.1 Datasets

The experiments have been conducted using two recent publicly available datasets that provide raw network captures.

IoT23 [14] is a dataset containing benign and malicious IoT network traffic divided into 23 scenarios. Three of them consist of benign traffic captured from real IoT devices in a smart home environment. The authors then created 20 malicious scenarios by uploading different attack instances to a RaspberryPI present in the environment. The captures last 24 h, except in cases where the number of generated packets increased too rapidly.

The second dataset is IoTID20 [16], that also considers devices of a typical smart home scenario, together with smartphones and laptops. In addition, also smartphones and laptops were connected to the network during This dataset includes 9 kinds of attacks (some of them are not in IoT23), such as various categories of DoS and DDoS attacks, ARP spoofing and operating system scanning.

4.2 Graph Neural Network Training

As previously mentioned, the GNN methods we have considered for anomaly detection, were originally employed by their authors in a *transductive* setting: there is a single graph, whose structure is known a priori, and the method is used to predict the class of some nodes (i.e. whether they are normal or anomalous) given a disjoint set of nodes which are known to be normal.

In our case, that setting is not appropriate: each graph correspond to a snapshot of observed network traffic, and snapshots captured at different times will very likely have a different structure. Thus, we had to modify the training procedure to a more conventional inductive setting, where the network learns a function that is independent of the graph structure, and then this function is used to classify the nodes of new graphs, that are different from the ones seen at training time. Notice that the conventional technique used for inductive learning, i.e. applying the optimization algorithm to fixed-size batches randomly sampled from the training set, was not directly applicable here: the GNN layers and the loss functions (as described in Sect. 3) assume that the structure of the graph (the adjacency matrix A) is provided, and all the nodes of the graph are given in input. So, we have used a different, entire graph from the training set at each learning step; the training set was made of graphs consisting only of normal nodes, while the validation set also contains abnormal nodes. The training and validation procedure is repeted different times by randomly changing the order of the input graphs sequence in order to prevent the GNNs from specialyzing on a particular order of graphs.

In more datails, the dataset IoT23 has been used to train and validate the GNNs, for each representation 10,041 graphs have been used for the training set and 9,227 for the validation, while 9,221 graphs have composed the test set. The graphs have been extracted from approximately 1,400,000 flows of which 1,000,000 are benign and 200,000 malicious. The dataset IoTID20 has be used only for testing the capability of the GNNs to generalize on data collect from a similar IoT context, but with different devices and network attacks. In particular, from IoTID20 we have extracted 209 graphs for each representation, starting from 108,983 malicious and 14,753 benign flows. For both the datasets the duration of a temporal snapshot used to generate the graphs has been set to 2.5 min. This choice has been mostly driven by memory constraints during the training since we needed to load the whole graphs on the GPU.

For the sake of clarity, we provide some details about the hyperparameters and the training parameters for each considered method. Regarding DOMINANT, we used an encoder with three layers and a feature reconstruction decoder with one layer. The hidden layers of the encoder had 28 nodes, and the dropout probability was 0.3. During loss and anomaly score computation, we set the value of α to 0.8 to balance the features reconstruction error and the structural reconstruction error. We optimized the GNN using ADAM with a learning rate of 0.0005. The autoencoder was trained for 100 epochs, and we set the patience level to 20. For OCGNN, we opted for a 2-layer GraphSAGE encoder, as it is known to work effectively in an inductive context. The hidden layer consists of 28 neurons. We set the contamination factor $1-\beta$, which denotes the proportion of nodes allowed to stay outside of the hyper-sphere, to 0.2. For optimization of the encoder, we used the AdamW optimizer with a learning rate of 0.0003 and decay coefficient. The model was trained for 50 epochs with a patience of 10. Finally, in the case of CONAD, we set the number of encoder layers to 3, while the number of decoder layers responsible for reconstructing features is set to 1. The hidden encoder layer is had 28 neurons, while the output layer had 14. We set the loss margin m to 0.5, and we introduced a percentage of artificial anomalies in the augmented graphs equal to 0.2% of the total number of nodes in the input graph. We used ADAM as optimizer, with a learning rate of 0.0005. The autoencoder was trained for 50 epochs with a patience of 10.

4.3 Results

In Table 1 we have reported precision, recall and F-score for all the combinations of GNN and representations previously mentioned. Since our system is an anomaly detector, we considered as positive the anomalous nodes (e.g. a true positive was a node representing an attack that was labeled as anomalous by the system).

From Table 1 it is possible to note that among the three representations, the extended TDG is performing significantly worse than the others on IoT23, the dataset used to train the GNNs. On the other hand, the results on IoTD20 may lead to a different conclusion, indeed, this representation seems to provide a good generalization capability to GNNs. But, considering the compostion of the two dataset, where the percentage of abnormal traffic in IoTD20 (about 87%) is considerably higher than in IoT23 (about 29%), we can reasonably conclude that the GNNs are specilized on IoT23 resulting in a large number of false positives on IoTID20. An evidence of that is the very high recall achieved by all the GNNs on IoTD20, while the precision is close to percentage of malicious flows. Unfortunately, this is a common problem when an anomaly detection system trained on a given scenario is moved on a different one. This is the only considered representation where graph nodes represent communication endpoints instead of communication flows. Also, note that extended TDGs include a lot of network-related features in the nodes, but they fail to bring any benefit to the detection task.

The two remaining representations, similarity graphs and traffic trajectory graphs, have similar performance, but the first one is slightly better in general. Remember that in similarity graphs, node adjacency is determined by the similarity of the communication flow characteristics, while in traffic trajectory graphs it depends only on the sharing of a common endpoint. Only while using OCGNN, we get a significant drop of recall, that can be blamed to difficulty in adapting this GNN to face the problem at hand. We suspect that similarity graphs are better suited at capturing the occurrence of similar anomalous behavior by several devices that are not immediately connected to each other, and this may help for many kinds of attacks.

Looking at the GNNs, the one that achieved the best performance in the average is DOMINANT. In particular, while working with similarity graphs, DOMINANT shows the best F-score on both the datasets, therefore it has the best trade-off between precision and recall. The results suggest that a classical graph autoencoder architecture, according to which DOMINANT has been

Table 1. Performance of the considered GNN and representations, in terms of precision (Prec), recall (Rec) and F-Score, evaluated over the test sets of both IoT23 and IoTID20 datasets.

GNN	Representation	IoT23			IoTID20		
		Prec	Rec	F-Score	Prec	Rec	F-Score
CONAD	Extended	0.4815	0.5542	0.5153	0.8502	0.9953	0.9171
	Similarity	**0.6807**	0.6725	0.6766	0.8699	0.8433	0.8564
	Trajectory	0.6756	0.7010	0.6881	0.8652	0.8441	0.8545
DOMINANT	Extended	0.4937	0.5399	0.5158	0.8466	**0.9955**	0.9168
	Similarity	0.6748	0.7324	**0.7024**	**0.8896**	0.9951	**0.9394**
	Trajectory	0.6137	**0.7814**	0.6875	0.8735	0.8855	0.8794
OCGNN	Extended	0.4857	1.0000	0.6539	0.8491	1.000	0.9184
	Similarity	0.5963	0.7268	0.6551	0.7154	0.2026	0.3158
	Trajectory	0.5000	1.0000	0.6667	0.8823	1.000	0.9375

designed, is more suitable for the task at hand. CONAD also is based on a graph autoencoder, but its use of manually designed models of anomalies introduces a bias in the system, that may lead to a decrease in performance.

Finally, OCGNN seem to be slightly behind the other two methods. OCGNN does not try to optimize reconstruction error as the other two methods do, but attempts to project the nodes into a space where the normal nodes reside in a possibly small hyper-sphere. The fact that on the test set this network shows a large difference between precision and recall appears to be a consequence of the fact that the optimal position and radius of this hyper-sphere may change when the graph structure changes, and thus this algorithm is less performant in an inductive setting than it would be in its original transductive usage.

5 Conclusions

In conclusions, in this paper we have faced the problem of detecting abnormal communication patterns in IoT networks using Graph Neural Networks. To this purpose, we have selected from the state of the art three different ways of representing network traffic in terms of graphs, and three graph-based anomaly detection algorithms. The resulting 9 combinations have been experimentally evaluated using two recent datasets composed of real IoT network traffic. proposed in the context of network analysis. The results that we have obtained in the experiments are encouraging. Further analysis is required to assess the reliability and robustness as well as the generalization capability of GNN in this context.

References

1. Abbasi, M., Shahraki, A., Taherkordi, A.: Deep learning for network traffic monitoring and analysis (NTMA): a survey. Comput. Commun. **170**, 19–41 (2021). https://doi.org/10.1016/j.comcom.2021.01.021
2. Aouini, Z., Pekar, A.: Nfstream: a flexible network data analysis framework. Comput. Netw. **204**, 108719 (2022)
3. Churcher, A., et al.: An experimental analysis of attack classification using machine learning in IOT networks. Sensors **21**(2), 446 (2021)
4. Deng, A., Hooi, B.: Graph neural network-based anomaly detection in multivariate time series. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, pp. 4027–4035 (2021)
5. Ding, K., Li, J., Bhanushali, R., Liu, H.: Deep anomaly detection on attributed networks. In: Proceedings of the 2019 SIAM International Conference on Data Mining, pp. 594–602. SIAM (2019)
6. Fahim, M., Sillitti, A.: Anomaly detection, analysis and prediction techniques in IOT environment: a systematic literature review. IEEE Access **7**, 81664–81681 (2019). https://doi.org/10.1109/ACCESS.2019.2921912
7. Iliofotou, M., Pappu, P., Faloutsos, M., Mitzenmacher, M., Singh, S., Varghese, G.: Network monitoring using traffic dispersion graphs (TDGs). In: Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement, pp. 315–320 (2007)
8. Koroniotis, N., Moustafa, N., Sitnikova, E., Turnbull, B.: Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: bot-IOT dataset. Future Gen. Comput. Syst. **100**, 779–796 (2019)
9. Lo, W.W., Layeghy, S., Sarhan, M., Gallagher, M., Portmann, M.: E-graphsage: a graph neural network based intrusion detection system for IOT. In: NOMS 2022–2022 IEEE/IFIP Network Operations and Management Symposium, pp. 1–9. IEEE (2022)
10. Lotfollahi, M., Siavoshani, M.J., Zade, R.S.H., Saberian, M.: Deep packet: a novel approach for encrypted traffic classification using deep learning. Soft Comput. **24**(3), 1999–2012 (2019). https://doi.org/10.1007/s00500-019-04030-2
11. Ma, X., et al.:: A comprehensive survey on graph anomaly detection with deep learning. IEEE Trans. Knowl. Data Eng. (2021)
12. Macas, M., Wu, C., Fuertes, W.: A survey on deep learning for cybersecurity: progress, challenges, and opportunities. Comput. Netw. **212**, 109032 (2022). https://doi.org/10.1016/j.comnet.2022.109032
13. Pacheco, F., Exposito, E., Gineste, M., Baudoin, C., Aguilar, J.: Towards the deployment of machine learning solutions in network traffic classification: a systematic survey. IEEE Commun. Surv. Tutor. **21**(2), 1988–2014 (2019). https://doi.org/10.1109/COMST.2018.2883147
14. Parmisano, A., Garcia, S., Erquiaga, M.J.: A Labeled Dataset with Malicious and Benign IOT Network Traffic. Stratosphere Laboratory, Praha, Czech Republic (2020)
15. The Guardian: DDoS attack that disrupted internet was largest of its kind in history, experts say. https://www.theguardian.com/technology/2016/oct/26/ddos-attack-dyn-mirai-botnet
16. Ullah, I., Mahmoud, Q.H.: A scheme for generating a dataset for anomalous activity detection in IoT networks. In: Goutte, C., Zhu, X. (eds.) Canadian AI 2020. LNCS (LNAI), vol. 12109, pp. 508–520. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-47358-7_52

17. Wang, X., Jin, B., Du, Y., Cui, P., Tan, Y., Yang, Y.: One-class graph neural networks for anomaly detection in attributed networks. Neural Comput. Appl. **33**, 12073–12085 (2021)
18. Xu, Z., Huang, X., Zhao, Y., Dong, Y., Li, J.: Contrastive attributed network anomaly detection with data augmentation. In: Advances in Knowledge Discovery and Data Mining: 26th Pacific-Asia Conference, PAKDD 2022, Chengdu, 16–19 May 2022, Proceedings, Part II, pp. 444–457. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-05936-0_35
19. Zheng, J., Li, D.: Gcn-tc: combining trace graph with statistical features for network traffic classification. In: ICC 2019–2019 IEEE International Conference on Communications (ICC), pp. 1–6. IEEE (2019)
20. Zheng, J., Zeng, Z., Feng, T.: Gcn-eta: high-efficiency encrypted malicious traffic detection. Secur. Commun. Netw. **2022**, 1–11 (2022)
21. Zola, F., Segurola-Gil, L., Bruse, J.L., Galar, M., Orduna-Urrutia, R.: Network traffic analysis through node behaviour classification: a graph-based approach with temporal dissection and data-level preprocessing. Comput. Secur. **115**, 102632 (2022)