DyGCN: Dynamic Graph Convolution Network-based Anomaly Network Traffic Detection

Yonghao Gu, Xiaoqing Zhang, Hao Xu, and Tiejun Wu

Abstract—Traditional abnormal network traffic detection methods only consider traffic statistical features and ignore structural relationships, which makes them difficult to detect advanced and complex attacks. Furthermore, the intrusion detection methods using static graph neural networks can't adapt to the dynamic changes of the network and have poor performance for anomaly network traffic detection. To solve these problems, we propose **DyGCN**(Dynamic Graph Convolution Network) for anomaly network traffic detection. Firstly, we represent the network at a certain moment with a graph using hosts as nodes and construct a graph stream according to the dynamic network. To aggregate information on the directed multi-graph and adapt to the dynamic changes of the network, we propose a dynamic graph model with structural learning and temporal learning for hosts. Secondly, in order to improve the effect of self-supervised learning, we put forward a contrastive learning method by analyzing the structural characteristics of normal traffic, where the negative edges are sampled far from the normal network pattern. In this way, we can reduce the sample deviation and improve the model's learning ability for normal traffic. Thirdly, we obtain the graph representation with a novel readout function to mine the anomalous substructures in the graph. Specifically, the graph representation is computed with edge embeddings and edge anomaly probabilities. Finally, with the graph representations, we use an anomaly detection model to judge the anomaly of the current network. Experimental results on two public datasets demonstrate that DyGCN outperforms traditional anomaly detection methods and state-of-the-art dynamic graph embedding methods.

Index Terms—Anomaly network traffic detection, Network attacks, Dynamic graph convolution network, Contrastive learning, Graph embedding

1 Introduction

With exponential growth in the size of computer networks and applications, the types of network attacks gradually increase, such as DDoS, Web Attacks, and Infiltration, which bring huge security risks to the enterprise. Network intrusion detection based on traffic information can quickly track malicious IPs and prevent the spread of intrusion in time, which has received considerable critical attention from both academia and industry.

In recent years, with the developing of advanced and stealthy attack technology, detecting attack at a single point or flow gets much harder and more inaccurate, as exemplified by the APT28 attack during the famous US presidential election2017[1]. As shown in Fig. 1, this cyber attack process can be summarized in three stages: preparation stage, attack stage, and post-attack stage. In the preparation stage, the CRU team carried out reconnaissance scans on the DNC network to identify its underlying infrastructure and purchased resources such as VPN services, domain names, and servers for follow-up attacks. In the attack stage, the spear-phishing attack was launched through the VPN server to steal the credentials of DNC users. With these login credentials, attackers installed the X-Agent malware on several computers to establish communication with the C&C server and collected keystroke capture data and screenshots. In the post-attack stage, they began to move laterally onto the DNC's server and then gather up documents, which were transmitted back to the middle server through an encrypted tunnel. The above three attack stages make the network structure of the DNC network boundary and intranet greatly changed compared with normal traffic. In order to reduce the risk of detection of command and control channels, the CRU team encrypts the communication

- *This work was supported in part by the Fundamental Research Funds for the Central Universities (Beijing University of posts and Telecommunications) for Action Plan under Grant 2021XD-A11-1; in part by the CCF-NSFOCUS Kun-Peng Scientific Research Fund under Grant CCFNSFOCUS202213; in part by the Opening Project of Guangdong Province Key Laboratory of Information Security Technology under Grant 2020B1212060078; and in part by the National Natural Science Foundation of China under Grant U20B2045.*

- *Yonghao Gu is with the Beijing Key Laboratory of Intelligent Telecommunications Software and Multimedia, School of Computer Science, Beijing University of Posts and Telecommunications, Beijing 100876, China, and also with the Guangdong Provincial Key Laboratory of Information Security Technology, Sun Yat-Sen University, Guangzhou 510275, China (e-mail: guyonghao@bupt.edu.cn).*
- *Xiaoqing Zhang is with the Beijing Key Laboratory of Intelligent Telecommunications Software and Multimedia, School of Computer Science, Beijing University of Posts and Telecommunications, Beijing 100876, China (e-mail: 1074913189@qq.com).*
- *Hao Xu is with the Beijing Key Laboratory of Intelligent Telecommunications Software and Multimedia, School of Computer Science, Beijing University of Posts and Telecommunications, Beijing 100876, China (e-mail: bzxuhao@163.com).*
- *Tiejun Wu is with the Nsfocus Technologies Group Co.Ltd., Beijing 100089, China(email:wutiejun@nsfocus.com).*
- *Yonghao Gu and Xiaoqing Zhang have equal contribution.*

1. https://arstechnica.com/information-technology/2018/07/from-bitly-to-x-agent-how-gru-hackers-targeted-the-2016-presidential-election/

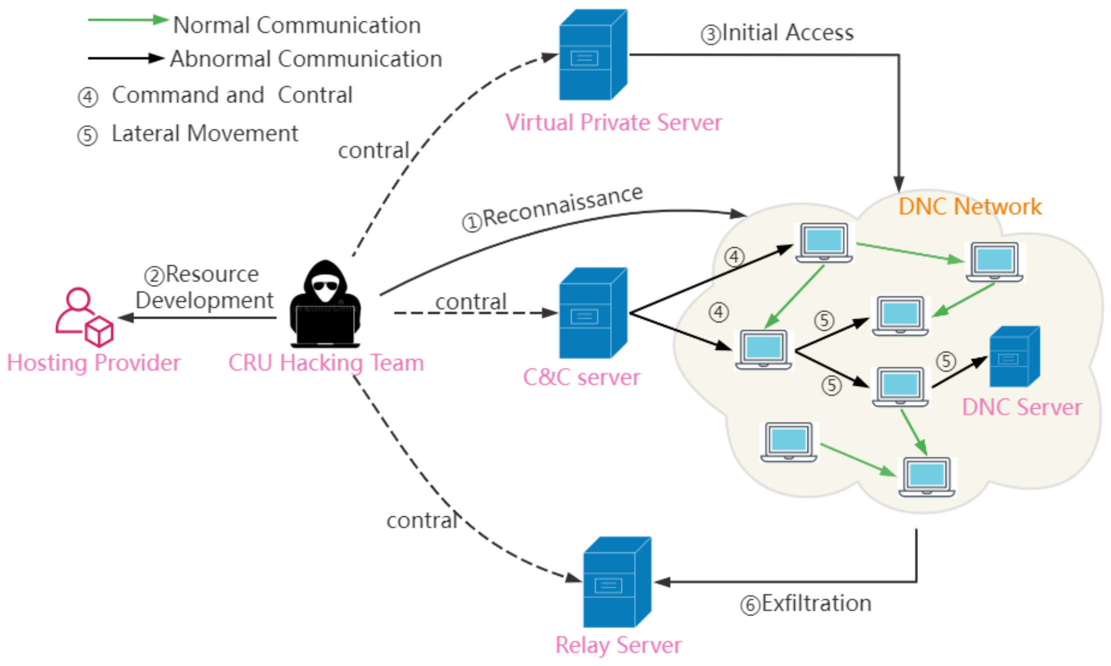


Fig. 1: APT28 Attack graph During the US Election

content and builds multi-channels to communicate with the DNC network with multi servers, which reduces the available information and maliciousness of CRU servers. However, the semantics of network structure between flows or IPs won't be affected. The traditional feature vector-based anomaly detection technologies[1–3] detect whether a single network flow or host is abnormal, which ignores the complex relationships among different hosts and is difficult to detect such advanced and stealthy attacks. Besides, these detection models are vulnerable to adversarial attacks formed by applying small but intentionally worst-case perturbations to the features[4] and require regular retraining to update the models.

Security analyzers try to correlate the attack steps and detect the attack chain as a whole or correlate different flows to detect the converged attack events. Fortunately, GNNs can capture the relational semantics between hosts(nodes) or between flows(edges) and have been used to detect anomalies in networks (graph)[5, 6]. However, the multi-stage characteristics of complex attacks cause continuous changes in the relationship between nodes and between edges in the network, and static GNNs cannot represent the dynamic changing semantics. Dynamic graph convolution networks can adapt to the dynamic changes of the network, but most are time-consuming for network traffic detection. Dynamic graph convolution networks-based anomaly network traffic detection methods have not been systematically studied. Besides, there is a lack of ground truth of attack data in the abnormal network traffic detection scenario, making supervised learning methods difficult to apply. Therefore, we will study how to detect abnormal traffic through a dynamic graph convolution network in a self-supervised way.

The existing dynamic graph convolution networks have the following problems so that they can't be directly used for anomaly network traffic detection:

- The existing dynamic graph convolution networks[7–9] are difficult to handle the dynamic changes of the network, including the appearance and disappearance of nodes/edges, and the changes in attributes of nodes/edges. Besides, the computational and storage costs are high when faced with large-scale graph-structured data.
- There exists large sample bias when performing negative sampling for the existing dynamic graph convolution networks[10–13]. The negative samples sampled might be normal that have appeared in the past or will appear in the future, which is common in networks where communication between nodes doesn't exist all the time. This bias caused by negative sampling will lead to inaccurate learning for normal traffic.
- The existing methods[14–16] that detect graph anomalies with graph embeddings have a poor effect because they do not distinguish the abnormality of different elements in the graph of malicious network which has both normal and abnormal flows.

In this paper, we propose a novel dynamic graph convolution network DyGCN for anomaly network traffic detection, which enables online learning and better detection performance. To address the problem (1), we introduce graph convolution networks to learn the structural features of the current network topology and introduce a temporal module to adapt to the changing network structure(In section 4.2). To address the problem (2), we put forward a contrastive learning strategy by analyzing the pattern of normal network communication behaviors(In section 4.3). To address the problem (3), we provide a readout function based on edge anomaly probabilities to calculate graph embedding(In section 4.4). The contributions in this paper can be summarized as follows:

- We propose an abnormal traffic detection method DyGCN which adapts to the characteristics of dynamic changes in the network and improves the real-time performance of existing dynamic graph neural network methods.
- We design a contrastive learning strategy based on the normal network traffic pattern. To reduce the bias caused by negative sampling, we select the negative samples according to the scores calculated with degrees of connectionless nodes.
- We present a readout function to calculate graph embeddings through edge anomaly probabilities. To detect abnormal events effectively, we obtain the global representation of the graph by calculating the weighted sum of the edge embeddings.
- We conduct comprehensive experiments on two open-source datasets that demonstrate consistently superior performance for DyGCN over state-of-the-art methods in anomaly network traffic detection.

2 Related Work

We will study how to design a dynamic graph convolution network on the basis of self-supervised learning and graph embedding. Therefore, in this section, we introduce the latest work on dynamic graph anomaly detection in Section 2.1 and investigate the self-supervised learning strategies in Section 2.2. Graph embedding in dynamic graph anomaly detection is introduced in Section 2.3.

2.1 Anomaly Detection on Dynamic Graphs

Dynamic graph anomaly detection methods can be divided into statistical methods and deep learning methods.

Dynamic graph anomaly detection methods by statistical learning formulate rules for intrusion rely on expert experience without parameter learning. Yoon et al.[17] propose AnomRank which focuses on the 1st and 2nd-order derivatives of node scores to detect sudden changes in the importance of any node. Eswaran et al.[18] detect anomalies involving the sudden appearance or disappearance of large dense directed subgraphs. This method first samples k subgraphs of the original graph. Then it calculates the sum weights of edges in each subgraph to generate a k-dimensional feature vector for anomaly detection. Bhatia et al.[12] propose MIDAS, which detects microcluster anomalies or suddenly arriving groups of suspiciously similar edges in edge streams. CMS data structure is used to approximately maintain the number of edges s_{uv} for all node pairs in constant memory. The anomaly score is expressed as a chi-square value between the theoretical and observed value of s_{uv}. The statistical methods are simple and efficient but are limited to specific anomaly types and difficult to detect unknown attack types.

Dynamic graph anomaly detection methods by deep learning integrate graph neural networks(GNNs) and recurrent neural networks (RNNs) to detect anomalies from the structure and temporal trends. Cai et al.[19] detect anomalous edges in dynamic graphs by leveraging GCN and Sortpooling layers which extract the edge representation from the subgraph centered on the target edge. Besides, Gated Recurrent Units (GRUs) are used to capture the temporal information for anomaly detection. Sankar et al.[20] present a Dynamic Self-Attention Network (DySAT), which computes node representations through joint self-attention along the two dimensions of the structural neighborhood and temporal dynamics. To capture the topological and hierarchical properties of graphs, Liu et al.[21] extract subgraphs at multiple scales using k-core decomposition[22], and the k-core subgraphs can be seen as the evolution process of the original graph. Global representation of the graph is captured by GCN and LSTM based on the k-core subgraphs. Compared with directly learning the graph embeddings generated from GNNs and RNNs, Pareja et al.[9] focuses on the model itself and propose EvolveGCN, which uses RNN to evolve the GCN parameters at every timestamp. Dynamic graph convolution networks are effective for anomaly detection problems but have not been systematically studied in the area of anomaly network traffic detection. There is only one relevant paper[9] which put forward a dynamic unsupervised representation learning framework called Dynamic-DGI (dynamic deep graph infomax). Firstly, this method exchanges the roles of nodes and edges in the original graph to obtain a line graph with edges as the basic elements, which requires an expensive computation cost. Then GCN is used on the original graph and the line graph respectively to obtain node embeddings and edge embeddings. Finally, the global representation of the graph is learned with the mutual information maximization approach[23]. This research doesn't consider the dynamic changes of the node set in the network.

2.2 Self-supervised Learning

The existing self-supervised learning strategies on graphs can be divided into two types: contrastive learning and generative learning[24]. Contrastive learning learns the representation of samples by comparing the positive samples and negative samples in the feature space. Compared with generative learning which needs to reconstruct sample details, contrastive learning only needs to learn the distinction between positive and negative samples in the feature space and pay more attention to abstract semantic information than details. The key to contrastive learning is how to define the positive and negative samples. Guo et al.[9] generate negative graphs by randomly perturbing the edges to change the structure of the original graphs. Graph embeddings are learned by maximizing mutual information between local information(node embedding) and global information(graph embedding). Chu et al.[15] adopt four graph augmentation techniques to obtain the positive samples, including node dropping, edge perturbation, attribute masking, and subgraph sampled by random walking. Negative samples are selected by curriculum learning. In addition to the graph sampling, most of the methods perform positive and negative sampling at the edge level according to whether the nodes are connected[19–21]. In a dynamic graph, nodes that are not connected at the current moment may be connected in the past or in the future, resulting in difficulty in model learning.

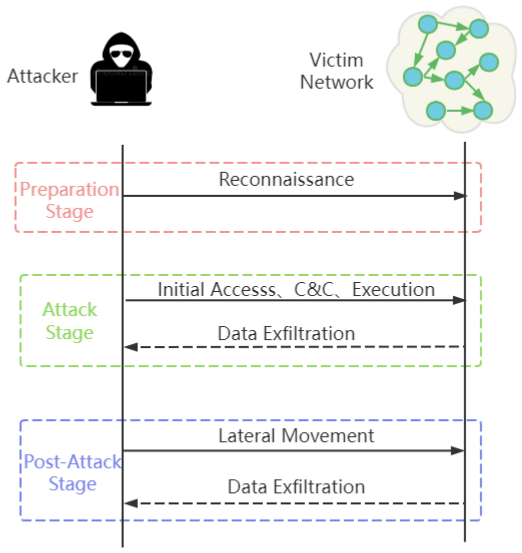


Fig. 2: Adversary Model of DyGCN

2.3 Graph Embedding

Subgraph representation learning[25] can't obtain the representation of the whole graph so it can't be used for graph classification and graph anomaly detection. Therefore, the graph embedding in this paper refers to global graph embedding. The graph kernel-based methods[13] are less versatile due to the use of custom features. Narayanan et al.[26] propose Graph2Vec by extending the Doc2Vec[27] to graph-level representation learning. Graph2Vec treats the entire graph as a document and the subgraphs of each node as words to learn the graph representation. This method uses random walking to generate subgraphs, which is time-consuming and can not meet the real-time requirements of abnormal traffic detection. Graph representations are mostly obtained with a readout function based on node embeddings or edge embeddings. The readout function can be implemented by pooling operation[15], sum operation[28], or mean operation[9]. In dynamic network anomaly detection, not all nodes and edges in the graph are abnormal, therefore we should try to avoid the influence of normal nodes and edges when computing graph embeddings.

3 PROBLEM DEFINITION

The adversary model of DyGCN is shown in Fig. 2. The attack is initiated by the attacker and contains three stages. The reconnaissance was performed to gather information about the victim network. In the attack stage, the attack is trying to access the victim network and build C&C communication to deliver malware which later is executed to collect useful data. With the controlled hosts, the attacker will move laterally to other hosts to find the target and collect information. In the whole attack process, the attacker often uses techniques to avoid detection of the victim network, including uninstalling/disabling security software or obfuscating/encrypting data and scripts. However, the attack traces left by the attacker in the structure are not affected. The attack traces contain malicious communication between multiple hosts, reflecting the attack intention.

The primary notations used in this paper are summarized in Table 1. We represent the hosts in the network as nodes and the network session as directed edges. The network can be represented with a directed multi-graph and changes dynamically over time. Let T be the maximum timestamp, and a graph stream $\mathbb{G}$ takes the form of $\{G^t\}_{t=1}^{T}$, where each $G^t = (V^t, E^t)$. An edge $uv \in E^t$ means that the network session is sent from node u to node v at the time t. $F^t \in \mathbb{R}^{m \times c}$, where m is the size of E^t and c is the feature dimension. $A^t \in \mathbb{R}^{n \times n}$, where $A^t[i][j] = w$, and the size of the node set is n at time t. $A^t_{in} \in \mathbb{R}^{n \times m}$, where $A^t_{in}[i][j] = 1$ if the i-th node is the destination of the j-th edge at time t, otherwise $A^t_{in}[i][j] = 0$. $A^t_{out} \in \mathbb{R}^{n \times m}$, where $A^t_{out}[i][j] = 1$ if j-th edge is from the i-th node at time t, otherwise $A^t_{out}[i][j] = 0$.

According to the above definitions, we model the network as a dynamic graph and execute anomaly detection on the dynamic graph to determine whether the graph snapshot is normal at the current moment.

Table 1: Primary Notation Definition

Notation	Definition
G^t	graph snapshot
V^t	node set
E^t	edge set
A^t	node adjacency matrix
A^t_{in}	association matrix between nodes and incoming edges
A^t_{out}	association matrix between nodes and outgoing edges
W^t	weight matrix
F^t	edge attribute matrix
H^t	node temporal embeddings
X^t	node structural embeddings

4 METHODOLOGY

In this section, we introduce the proposed method DyGCN. The framework of DyGCN is introduced in Section 4.1. From Section 4.2 to Section 4.5, we concretely introduce the four main components of the DyGCN framework in detail.

4.1 DyGCN Framework

The overview of DyGCN framework is illustrated in Fig. 3. DyGCN consists of four parts, namely spatio-temporal node embedding, normal traffic pattern-based contrastive learning, edge anomaly probability-based graph embedding, and anomaly detection. Firstly, spatial features X^t are extracted by aggregating flow attributes and used for temporal feature learning through a temporal module. Then we carry out negative sampling according to the analysis of the normal traffic pattern, and the constructed negative edges $\tilde{E}^t$ will be used for contrastive learning together with the positive edges E^t. To find malicious events, we propose a novel readout function to calculate the graph representations g^t based on the edge anomaly probability vector p^t and edge embeddings. p^t is computed with a MLP network $f(\cdot)$. Finally, anomaly detection is performed with the graph representations.

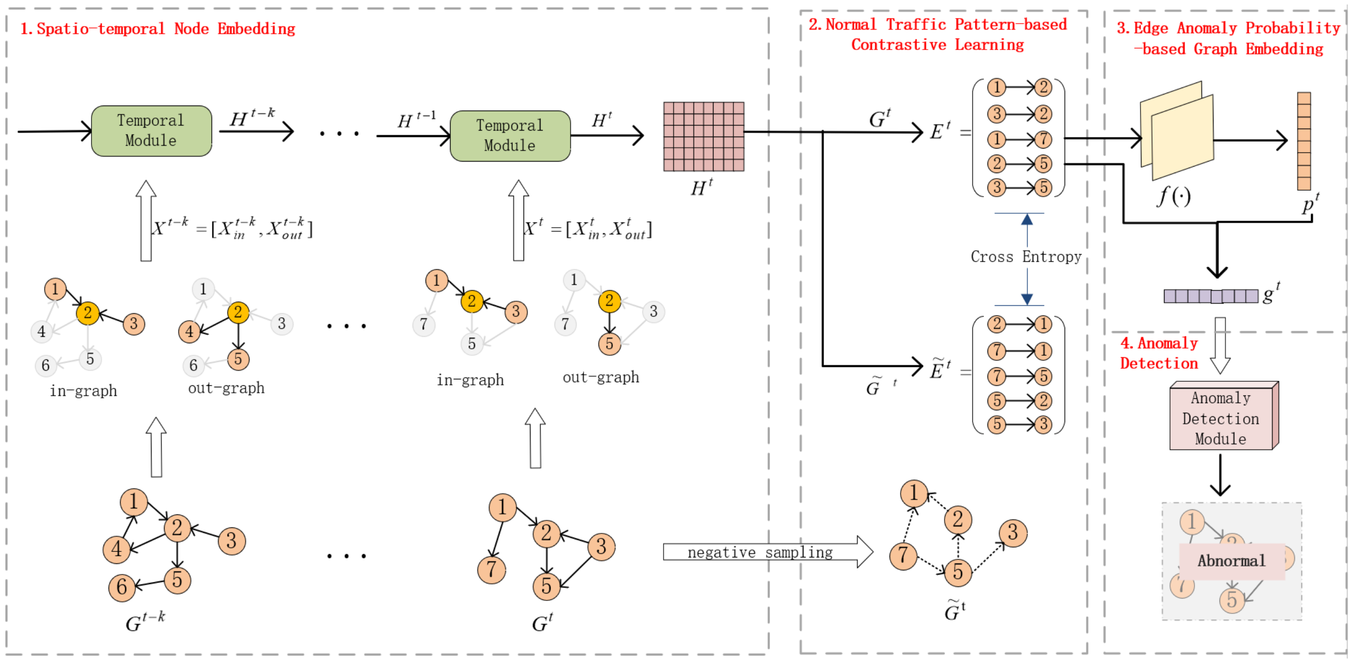


Fig. 3: Overall framework of DyGCN

4.2 Spatio-temporal Node Embedding

4.2.1 *Structural Learning*

During complex network attacks, attackers need to communicate with victims to convey malicious commands. Encoding, obfuscation, encryption, and other technologies for communication content are frequently used to reduce the risk of detection of the C&C channel, which will affect the attributes of a single flow and node, but won't affect the semantics of structure among multiple flows or IPs. In complex network attacks, attackers control normal hosts through various means such as phishing and vulnerability scanning and further move laterally through the controlled hosts to build a botnet or search for valuable servers for data theft or device destruction, which can affect the normal network structure. Therefore, we use GCN for structural learning on normal networks to find the anomalous substructure and compensate for the missing semantics in communication. Besides, the direction of edges in the network indicates the order and dependency of communication, so the communication semantics generated by the flow direction needs to be considered. We perform structural learning on the directed multi-graph constructed according to network communication behaviors. The structural learning process is shown in Fig. 4, we consider the incoming and outgoing edges of nodes separately and convert the directed graph into two undirected graphs denoted by in-graph and out-graph respectively. The process of features aggregation is defined as Eq.(1) and Eq.(2). Finally, the structural features of nodes can be represented by concatenating X_{in}^t and X_{out}^t through Eq.(3). The information of multi-order neighbors can also be aggregated by stacking multiple GCN network layers.

$$X_{in}^t = \sigma(\hat{A}_{in}^t F^t W_{in}^t), \hat{A}_{in}^t = \hat{D}_{in}^t A_{in}^t, \tag{1}$$

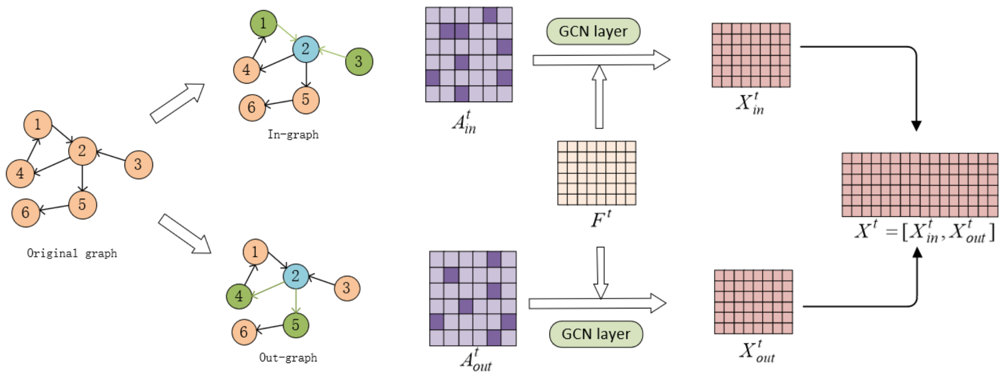


Fig. 4: Node embedding with GCN. We split the original graph into an in-edge graph and an out-edge graph. Specifically, for blue node 2 in the in-edge graph, we only consider its incoming edge information from green node 1 and node 3. While in the out-edge graph, the information sent to node 4 and node 5 is concerned.

$$X_{out}^t = \sigma(\hat{A}_{out}^t F^t W_{out}^t), \hat{A}_{out}^t = \hat{D}_{out}^t A_{out}^t, \quad (2)$$

$$X^t = Concat(X_{in}^t, X_{out}^t), \quad (3)$$

where X_{in}^t and X_{out}^t respectively represent the results of graph convolution on in-graph and out-graph at time t. X^t represents the final node embeddings at time t. W_{in}^t and W_{out}^t represent the parameter matrices of the GCNs. $\hat{A}_{in}^t$ and $\hat{A}_{out}^t$ are obtained by random walking normalizing A_{in}^t and A_{out}^t. $\hat{D}_{in}^t$ and $\hat{D}_{out}^t$ are diagonal matrices representing the in-degree matrix and out-degree matrix of nodes respectively, in which diagonal element is $\hat{D}_{ii} = \Sigma_j A_{ij}$. σ is the activation function.

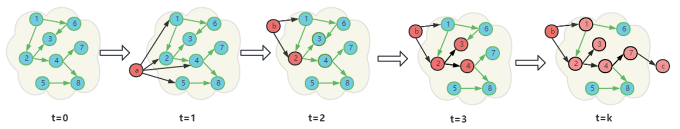


Fig. 5: Structural changes caused by APT28 attacks during the US election. The red nodes represent malicious hosts and the blue nodes represent normal hosts.

4.2.2 *Temporal Learning*

The network structure will change over time, including the addition and disappearance of nodes, the addition and disappearance of edges, and changes in edge direction, as shown in Fig. 5. The attack servers a, b, and c communicate with the target network, causing changes in the target network structure. As the attack deepens, the impact of the attack on the target network structure becomes more and more apparent, and more normal nodes are controlled to perform malicious communications. To associate the structural effects generated by attacks at different stages, we introduce LSTM for temporal feature learning of nodes. Since the network graph is dynamically changing which can be seen from the Fig. 6 that nodes will appear or disappear with time, and some nodes in the current graph may be missing in the historical graphs. For example, the malicious node c joined the network communication graph at t=4 according to the Fig. 5, LSTM cannot handle the problem that the size of the node set is not fixed. To solve this problem, we supplement the missing historical state of nodes by recording the features and IPs of nodes in the last k time steps and aligning the historical node set with the current node set. Specifically, we set the missing historical state to a zero vector for nodes in the current node set. The historical state sequence of node i at time t can be represented as $\{h_i^j | j \in [t-p, t-1], h_i \in H_i\}$. The temporal dependencies learning process of LSTM can be defined as Eq.(4) - Eq.(9).

$$f^t = \sigma(W_f[H^{t-1}, X^t] + b_f), \quad (4)$$

$$i^t = \sigma(W_i[H^{t-1}, X^t] + b_i), \quad (5)$$

$$\tilde{c}^t = tanh(W_c[H^{t-1}, X^t] + b_c), \quad (6)$$

$$c^t = f^t * c^{t-1} + i^t * \tilde{c}^t, \quad (7)$$

$$o^t = \sigma(W_o[H^{t-1}, X^t] + b_o), \quad (8)$$

$$H^t = o^t * tanh(c^t), \quad (9)$$

where W and b are the parameters of LSTM, H^{t-1} is the hidden states of nodes at time t-1, c^{t-1} is the cell state at time t-1, X^t is the structural embeddings of nodes at time t, f^t is the forget gate at time t, i^t is the input gate at time t, o^t is the output gate at time t, $\tilde{c}^t$ is the candidate cell state at time t, c^t is the cell state at time t, and σ is the activation function.

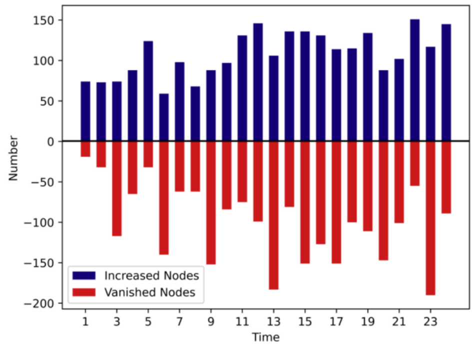


Fig. 6: The dynamic change of node set size, which observed on the CIC-IDS 2017 dataset. Each graph contains 1000 edges, and this figure shows the number of increased and vanished nodes in the first 24 snapshots.

4.3 Normal Traffic Pattern-based Contrastive Learning

We perform contrastive learning by negative sampling. It is not appropriate to construct the negative dataset only based on the unconnected nodes in the current graph which may have been connected before or will connect later. For example, in Fig. 5, the edge (v_6, v_3) is sampled as negative at t=3, but was positive at t=2, this bias caused by negative sampling is not conducive to model learning. The negative edges should be consistent with the real network and different from the normal traffic distribution as far as possible. We analyze the normal networks of experimental datasets and find that there are always some nodes with large degrees in the normal network at different moments, which are called centers. The nodes connected to the centers are named as edge nodes. By analyzing the normal network at different timestamps, we find that there is almost no connection between centers themselves and also among edges themselves which can be seen from Fig. 7. The centers can be regarded as servers, and the edge nodes can be treated as clients. The servers are usually independent and responsible for receiving and responding to requests from clients. There is less communication between clients which mainly communicate with the servers. According to the above characteristics of the normal network, we define the negative sampling scores between the unconnected nodes as Eq.(10). In this way, we can choose the top K negative edges to construct the negative dataset $\tilde{E}^t$. K is selected according to the size of the positive dataset. Positive dataset E^t are composed of the existing edges in the current graph.

$$s_{uv} = \frac{d_u + d_v}{|d_u - d_v| + \delta}, \quad (10)$$

where s_{uv} represents the negative sampling score of the edge uv that there is no connection from source node u to the destination node v. d_u and d_v are the degrees of the source node and the destination node respectively. δ is to prevent the denominator from being zero.

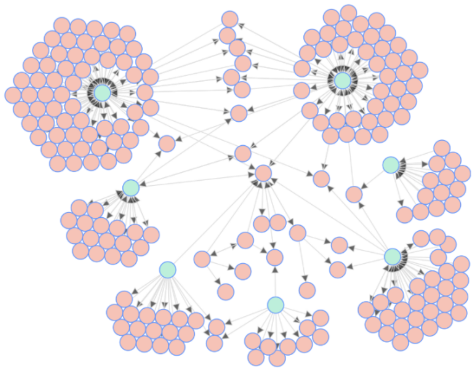


Fig. 7: Normal network where the green nodes are centers.

In order to verify the above analysis, we calculated the average number of occurrences of negative edges sampled by a random strategy and our strategy in the current graph at the historical and future networks. To reduce the computation, we randomly selected five graphs on the datasets CIC-IDS 2017 and CSE-CIC-IDS 2018 for computation. The result is shown in Table 2. We can see that the negative edges sampled by random negative sampling appear more frequently in the history and future than DyGCN negative sampling, which will cause greater deviation.

Table 2: Average number of historical and future occurrences of negative edges

	CIC-IDS 2017	CSE-CIC-IDS 2018
Random negative sampling	4301	5
DyGCN negative sampling	**44**	**1**

According to the node embeddings H^t calculated by structural and temporal learning, we concatenate the source node embedding and destination node embedding of the target edge to obtain the edge embedding. Then the edge embeddings are used for classification with an MLP network. The training phase of DyGCN is shown in Algorithm 1. The loss is calculated with:

$$L^t = -\frac{1}{N}\sum_{i=1}^{N}(y_i log f(e_i) + (1-y_i) log(1-f(e_i))), \quad (11)$$

where L^t is the loss calculated on G^t, $y_i = 0$ if the i-th edge is negative, otherwise $y_i = 1$, N is the size of $E^t \cup \tilde{E}^t$, $f(\cdot)$ is realized with an MLP network, e_i is the embedding of i-th edge.

4.4 Edge Anomaly Probability-based Graph Embedding

Complex network attacks often involve communications between multiple hosts, such as port scanning, vulnerability scanning, and network phishing during the preparation phase to obtain access credentials of the target network. During the attack phase, the attacker establishes command

Algorithm 1 Training procedure of DyGCN

```
Input: Training set $\mathbb{G} = \{G^t\}_{t=1}^{T}$, number of training
    epochs: $I$.
Output: Model parameters: $\theta$
    Randomly initialize the parameters of GCN model,
    LSTM model, and MLP model.
 1: for $i \in 1, 2, ..., I$ do
 2:     for graph $G^t = (V^t, E^t) \in \mathbb{G}$ do
 3:         Calculate node embedding matrix $H^t$ via
 4:         Eq.(1)-Eq.(9);
 5:         Calculate $S = \{s_{uv} | u, v \in V^t, uv \notin E^t\}$ via
 6:         Eq.(10);
 7:         Construct $\tilde{E}^t = \{uv | s_{uv} \in top-k(S)\}$;
 8:         for $uv \in E^t \cup \tilde{E}^t$ do
 9:             Calculate the embedding of edge $uv$ by
10:             $e_{uv} = Concat(h_u^t, h_v^t)$;
11:         end for
12:         Calculate loss $L^t$ via Eq.(11);
13:         Back propagation and update the parameters $\theta$;
14:     end for
15: end for
```

and control channels with several normal hosts using credentials. In the post-attack phase, the attacker uses the controlled hosts to penetrate the network, discover and control more hosts, steal valuable data, or damage equipment. Compared with node anomaly detection or edge anomaly detection to find malicious hosts or flows, graph anomaly detection with multiple nodes and edges can associate hosts and flows to discover the malicious attack activity. Detecting the complex attacks in the network can be translated into detecting anomalous substructures in the graph. To perform global structural feature learning, we calculate graph embedding representation based on edge embeddings. To improve the ability of the model to identify abnormal substructures, we propose a novel readout function to calculate graph representation based on edge anomaly probabilities. The edge anomaly probabilities are obtained by calculating $f(e_i)$. $f(e_i)$ is a probability that the i-th edge is positive. In the test phase, we first calculate the edge embedding e_i according to the node embeddings. Then edge anomaly probability p_i can be obtained for every edge in the graph with Eq.(12). Finally, we get the graph embedding g^t with e_i and p_i according to the readout function defined by Eq.(13). g^t fuses the information of all edges selectively according to the probabilities. The selected edges with higher abnormal probabilities have greater weights in g^t, which can effectively extract the abnormal information from the graph.

$$p_i = 1 - f(e_i), \quad (12)$$

$$g^t = \sum_{i=1}^{m} p_i e_i, s.t. \sum_{i=1}^{m} p_i = 1, \quad (13)$$

4.5 Anomaly Detection

We perform anomaly detection with OCSVM[26] which using the graph embeddings as input. The optimization target of OCSVM can be expressed with Eq.(14). The first part minimizes the structural risk by minimizing the radius

R of the hypersphere. The second part constructs a slack variable ζ_i with a penalty coefficient C to minimize the empirical risk, which makes the model have the ability of fault tolerance. The empirical risk is 0 when all of the samples are inside the hypersphere.

$$\begin{gathered} min_{R,o,\zeta} V(R) + C \sum \zeta_i, \\ s.t. ||x_i - o||_2 \leq R^2 + \zeta_i, \zeta_i \geq 0, \forall i = 1, 2, 3, ..., m, \end{gathered} \tag{14}$$

where $V(R)$ represents the volume of the hypersphere, x_i represents the feature vector of the i-th sample, and o represents the center of the hypersphere.

OCSVM detects anomalies by establishing a hypersphere that contains all of the training graphs. A new graph will be predicted to be normal if it is inside the hypersphere, otherwise predicted to be abnormal. We can get the anomaly result of the current graph by getting the graph embedding as the input of the anomaly detection model which doesn't has to be OCSVM.

5 Experiments

To analyze model performance from different perspectives, we utilize three research questions to guide our experiments:

(RQ1) How is the performance of DyGCN compared to baseline methods on anomaly network traffic detection?

(RQ2) How efficient is DyGCN compared to the baseline methods for detecting abnormal network traffic?

(RQ3) Are the proposed contrastive learning strategy and graph embedding method in DyGCN effective?

5.1 Experimental Setup

5.1.1 Datasets

We conduct experiments on two network traffic benchmark datasets CIC-IDS 2017[2] and CSE-CIC-IDS 2018[3] to evaluate model performance.

CIC-IDS 2017 is generated by the Canadian Institute of Cyber Security. It simulates the most up-to-date common attack scenarios to generate malware traffic. The benign traffic is captured with the proposed B-Profile system.

CSE-CIC-IDS 2018 is collected from the collaborative project between the Communications Security Establishment (CSE) & the Canadian Institute for Cybersecurity (CIC). The normal traffic is captured in the same way as CIC-IDS 2017. The malware traffic is generated with the proposed M-Profile. The data file 2018-02-20 is used for experiments since other data files don't contain the IPs needed to construct graphs.

Both of the datasets provide csv files that contain 77-dimensional statistical flow features extracted with the CICFlowMeter tool. Compared with the CIC-IDS 2017, CSE-CIC-IDS 2018 covers a larger network. The traffic data needs to be preprocessed before being fed into the model. We can preprocess CIC-IDS 2017 and CSE-CIC-IDS 2018 uniformly as follows. Detailed statistics of the aforementioned datasets are shown in Table 3.

2. https://www.unb.ca/cic/datasets/ids-2017.html
3. https://www.unb.ca/cic/datasets/ids-2018.html

- Graph construction: Nodes in the graph are identified with IPs and directed edges are created according to the source IP and destination IP of each flow. The edge attributes are expressed by the flow features. We randomly generate fixed size(200-dim) vectors to represent node attributes which are not provided by the datasets.
- Normalization: The attributes of nodes and edges are normalized with linear normalization.
- Graph partition: The size of the graph snapshot can be determined according to the actual requirements, such as the period of the attack to be detected, the volume of the network traffic, the scale of the network, etc. According to the size of the dataset used in the experiments, we construct a graph stream $\mathbb{G}$ for every 1000 flows chronologically.
- Graph labeling: The graph is labeled as malicious when anomalous flow exists in the graph, otherwise labeled as benign.

Table 3: The introduction of datasets

Dataset	CIC-IDS 2017	CSE-CIC-IDS 2018
Node Number	19211	32541
Edge Number	2824000	7489000
Duration	5 days	1 day
Graph Number	2824	7489
Attack types	14	1

5.1.2 Baselines

OCSVM[29]: It's a traditional anomaly detection method that can't handle graph data. The input of the model can only be feature vectors of flows.

GCN[30]: It performs convolution on a static graph, and node embedding is obtained by aggregating neighbor information. Taking the whole static graph as input as well as node attributes.

GCN-LSTM[31]: It achieves dynamic graph modeling by fusing GCN and LSTM. GCN is used to generate node embeddings on each graph and LSTM is used to learn the temporal dependencies of nodes. Taking the dynamic graph as input as well as node attributes.

EvolveGCN[32]: It uses RNN to evolve the GCN parameters. There are two versions of EvolveGCN: EvolveGCN-H and EvolveGCN-O. EvolveGCN-H incorporates additional node embedding in the recurrent network. EvolveGCN-O focuses on the change of the structure. Taking the dynamic graph as input as well as node attributes.

Dynamic-DGI[9]: It evolves the graph embeddings with LSTM. The edge attributes are considered when performing graph convolution with GCN. Self-supervised learning with mutual information maximization on dynamic graphs is proposed. Taking the dynamic graph as input as well as node attributes and edge attributes.

DySAT[20]: It learns the contextual representations of nodes spatially and temporally based on the multi-head attention mechanism. Taking the dynamic graph as input as well as node attributes.

CTGCN[21]: It extracts K-core subgraphs of the original graph. In addition, two LSTMs are used to evolve the K-core subgraphs and the global graphs at different timestamps separately. Taking the dynamic graph as input as well as node attributes.

Table 4: The division of datasets

Dataset	Training set	Test set
CIC-IDS 2017	529	2290
CSE-CIC-IDS 2018	4600	2884

5.1.3 Experimental Design

The data in the training set should be normal and time-continuous when performing dynamic graph anomaly detection tasks. Therefore, for the CIC-IDS 2017, the training set is constructed from the data produced on Monday and the test set consists of the data from Tuesday to Friday. For the CSE-CIC-IDS 2018, we ignore the first 400,000 records which contain anomalies, the training set is constructed from the next 4.6 million records and the test set consists of the rest of the dataset. The ratio of normal and abnormal samples in the test set is about 1:1 for both datasets. The division results of datasets are shown in Table 4 where the value represents the number of split graphs.

To evaluate the performance of the model, we draw the ROC curve by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings and the area under the ROC curve can be expressed as AUC. The model performs well when AUC is high. We also calculate precision, recall, and F1 metrics corresponding to the optimal cut-point($max(TPR - FPR)$)[33] of ROC curve. The Top K measurement is used to analyze the robustness and generalization ability of the model. In order to compare the efficiency of different models, we calculate the average training time of a single epoch and the data preprocessing time.

5.1.4 Parameter Settings

For GCN and GCN-LSTM, we utilize two layers in GCN. For Dynamic-DGI, we utilize a single layer in GCN which is used for convolution on original graphs with IP as node and line graphs with flow as node. For DyGCN, two GCNs are used to aggregate in-edge information and out-edge information respectively. For EvolveGCN, DySAT, and CTGCN, we set the parameters according to the provided source code. Besides, for LSTM-based models, we split the graph dataset using a sliding window of size 5 and stride 1. All models are trained for 50 epochs. We adopt the Adam optimizer with a learning rate of 0.0001. The dropout rate is 0.2. Layer normalization is used in DyGCN. The dimension of node embeddings is set to 16 for DyGCN. We calculate the mean value of node embeddings as the graph representation except DyGCN and Dynamic-DGI.

5.1.5 Computing Infrastructures

All methods are implemented using PyTorch 1.9.0. All experiments are conducted on a computer server with NVIDIA GeForce RTX 3090 (24GB memory) GPUs, an Intel(R) Xeon(R) Silver 4210R CPU @ 2.40GHz, and 32 GB of RAM.

5.2 Anomaly Detection Performance(RQ1)

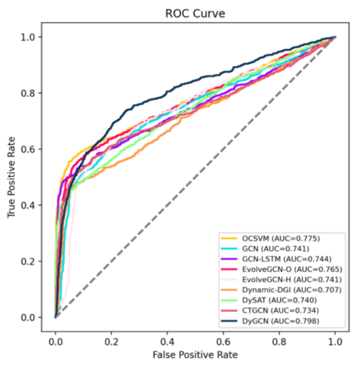
With the graph representations from dynamic graph convolution networks, we can perform anomaly detection through downstream models. The abnormal threshold is represented by the abnormal score corresponding to the optimal cut-point of ROC curve. The experimental results are shown in Table 5. The AUC of OCSVM is much better than GCN, especially on CSE-CIC-IDS 2018, which shows that only considering the structural information and ignoring the attributes of edges will reduce the performance of the model. Node temporal dependencies learning can improve the performance of GCN-based dynamic graph anomaly detection, which can be seen that GCN-LSTM performs better than GCN by introducing LSTM. DyGCN considers the attributes of edges when aggregating information from neighbors with GCN and learns the temporal dependencies of nodes with LSTM. EvolveGCN-O performs better than EvolveGCN-H. Compared to EvolveGCN-O, EvolveGCN-H adds node attributes EvolveGCN-H when learning the temporal dependencies of GCN weights, which introduces noise. DySAT conducts contrastive learning by random walks, encouraging nodes co-occurring in fixed-length random walks to have similar representations. While in the network, the length of random walks is relatively short and more obvious in the experimental datasets. CTGCN has the same problem when implementing contrastive learning by random walks. As can be seen from Table 5, DyGCN achieves gains of 1%-2% AUC and gains of 2%-6% F1 in comparison to the best baseline across all datasets. Furthermore, DyGCN achieves the optimal recall on both datasets which indicates that DyGCN can better capture abnormal activities occurring in the network.

To further analyze the anomaly detection ability of different models, we calculate precision and recall under Top K measurement that the fist K samples with the largest abnormal scores are judged as abnormal and the rest of the samples are judged as normal. The results are shown in Table 6. It can be seen that with the increase of K, the precision of all models decreases and the recall increases, indicating that more and more samples are predicted to be abnormal which results in increase in both false positives and true positives. The sample is easier to be judged as abnormal with a larger abnormal score. As the abnormal threshold decreases with the increase of K, the number of abnormal samples with small abnormal scores will increase which are difficult to detect. OCSVM and Dynamic-DGI have a better effect when K is relatively small, considering more traffic information than other models, and traffic characteristics are effective for identifying simple attacks. DyGCN doesn't perform best when K is small but gradually outperforms baselines as K increases, indicating that DyGCN can learn more discriminative features which can identify more abnormal samples than baselines. In addition, it also shows that the negative sampling strategy of DyGCN is more in line with the characteristics of attacks in real network scenarios.

We draw the ROC curve to evaluate the overall performance of models on both datasets. It can be seen from Fig. 8 that DyGCN outperforms baselines with the largest AUC. As the threshold increases, the performance of the

Table 5: Anomaly detection results

Model	CIC-IDS 2017 AUC	CIC-IDS 2017 Precision	CIC-IDS 2017 Recall	CIC-IDS 2017 F1	CSE-CIC-IDS 2018 AUC	CSE-CIC-IDS 2018 Precision	CSE-CIC-IDS 2018 Recall	CSE-CIC-IDS 2018 F1
OCSVM	77.49	91.13	55.65	69.1	87.74	78.29	82.59	80.38
GCN	74.07	82.50	51.52	63.43	70.66	66.94	76.49	71.40
GCN-LSTM	74.42	**91.74**	49.95	64.68	79.35	71.04	84.80	77.31
EvolveGCN-O	76.53	86.02	55.92	67.78	61.29	60.57	79.84	68.88
EvolveGCN-H	74.13	77.47	56.84	65.57	57.33	57.32	83.39	67.94
Dynamic-DGI	70.69	87.41	46.56	60.75	96.01	91.85	88.35	90.06
DySAT	73.96	89.54	44.81	59.73	57.04	55.14	81.98	65.93
CTGCN	73.39	87.80	50.23	63.90	83.48	75.08	79.91	77.42
DyGCN	**79.21**	77.77	**66.48**	**71.68**	**97.27**	**95.01**	**95.58**	**95.29**



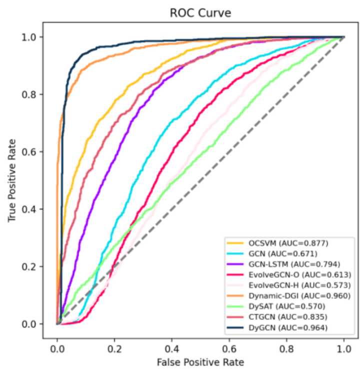


Fig. 8: The ROC curve of CIC-IDS 2017 (left) and CSE-CIC-IDS 2018 dataset (right)

DyGCN model gradually exceeds other models, indicating that the generalization ability of DyGCN is the best, which is consistent with the conclusion in Table 6 and Table 7.

5.3 Anomaly Detection Efficiency(RQ2)

We count the data preprocessing time and average training time per epoch for each GCN-based model on the CIC-IDS 2017 dataset under the same computing infrastructures. As illustrated in Table 8, GCN is highly efficient but can't process dynamic graphs and has poor performance in anomaly detection. Among the dynamic graph models, DyGCN has the shortest training time because the dynamic graph models except for DyGCN are trained at each timestamp according to the entire node set which is more time-consuming. Besides, the data preprocessing time of DyGCN is a little longer than GCN-LSTM, EvolveGCN-O, and EvolveGCN-H because DyGCN needs to calculate in-edge association matrix A_{in} and out-edge association matrix A_{out}. Dynamic-DGI is the most time-consuming model in data preprocessing due to the conversion from original graphs to line graphs. Random walking-based sampling adopted by DySAT and CTGCN is more complicated than the negative sampling strategy of DyGCN. Moreover, CTGCN needs to extract K-core subgraphs for each snapshot, and LSTM is used on both the K-core subgraphs and the global graph embeddings during the training phase.

5.4 Analysis for Contrastive Learning and Graph Embedding(RQ3)

We conduct two ablation experiments to evaluate the proposed methods about normal traffic pattern-based contrastive learning and edge anomaly probability-based graph embedding.

Table 6: Top K metrics on CIC-IDS 2017 dataset

Model	Precision@K 500	Precision@K 1000	Precision@K 1500	Precision@K 2000	Recall@K 500	Recall@K 1000	Recall@K 1500	Recall@K 2000
OCSVM	**96.40**	71.40	57.33	50.70	**44.26**	65.56	78.97	93.11
GCN	88.40	70.20	57.40	49.80	40.59	64.46	79.06	91.46
GCN-LSTM	96.20	68.90	56.20	49.55	44.17	63.27	77.41	91.00
EvolveGCN-O	93.00	71.10	58.07	50.80	42.70	65.29	79.98	93.30
EvolveGCN-H	84.40	70.30	58.00	50.25	38.75	64.55	79.89	92.29
Dynamic-DGI	89.40	64.40	55.40	49.50	41.05	59.14	76.31	90.91
DySAT	91.40	67.10	56.80	50.40	41.97	61.62	78.24	92.56
CTGCN	92.00	69.70	55.80	49.15	42.24	64.00	76.86	90.27
DyGCN	89.60	**75.20**	**60.53**	**51.10**	41.14	**69.05**	**83.38**	**93.85**

Table 7: Top K metrics on CSE-CIC-IDS 2018 dataset

	Precision@K				Recall@K			
Model	**500**	**1000**	**1500**	**2000**	**500**	**1000**	**1500**	**2000**
OCSVM	94.60	87.70	79.13	70.00	31.68	58.74	79.50	93.77
GCN	59.40	65.10	65.80	62.35	19.89	43.60	66.11	83.52
GCN-LSTM	78.80	76.70	73.73	67.95	26.39	51.37	74.08	91.02
EvolveGCN-O	48.60	59.50	61.67	60.25	16.28	39.85	61.96	80.71
EvolveGCN-H	48.00	55.30	57.53	57.35	16.08	37.04	57.80	76.83
Dynamic-DGI	**100.00**	**98.80**	89.73	72.85	**33.49**	**66.18**	90.15	97.59
DySAT	59.60	57.70	56.07	55.05	19.96	38.65	56.33	73.74
CTGCN	89.40	82.80	75.67	67.90	29.94	55.46	76.02	90.96
DyGCN	95.40	97.10	**95.00**	**74.30**	31.95	65.04	**95.45**	**99.53**

Table 8: Time efficiency of GCN-based models on CIC-IDS 2017 dataset

Model	Training time(s/epoch)	Data preprocess time(s)
GCN	6	118
GCN-LSTM	42	122
EvolveGCN-O	32	117
EvolveGCN-H	42	117
Dynamic-DGI	162	3501
DySAT	72	866
CTGCN	260	1972
DyGCN	16	148

Contrastive Learning We replace the negative sampling process of DyGCN with the degree distribution-based sampling strategy[8, 21] (DyGCN-Degree) to verify the effectiveness of our negative sampling strategy. The negative edges sampled on the basis of degree distribution have the same source nodes with positive edges, and the destination nodes of negative edges are sampled according to the degree distribution of all nodes in the network. As illustrated in Table 9, DyGCN outperforms DyGCN-Degree on the above two datasets, which indicates that the negative sample set obtained by the DyGCN-Degree may contain edges that appear in the past or in the future which will introduce sample bias, resulting in poor learning effect of the model.

Table 9: AUC scores of different contrastive learning strategies

	CIC-IDS 2017	CSE-CIC-IDS 2018
DyGCN-Degree	78.23	56.53
DyGCN	**79.21**	**97.27**

Graph Embedding We also average node embeddings as graph embedding for DyGCN(DyGCN-Mean) to evaluate the effect of the proposed edge anomaly-based graph embedding method. As illustrated in Table 10, DyGCN outperforms DyGCN-Mean on the above two datasets. The graph that contains malicious flow and benign flow should be predicted as abnormal while DyGCN-Mean will introduce more normal information than DyGCN for abnormal graphs, which leads to poor anomaly recognition. This has a greater impact on the CIC-IDS 2017 dataset as shown in Table 10.

Table 10: AUC scores of different graph embedding strategies

	CIC-IDS 2017	CSE-CIC-IDS 2018
DyGCN-Mean	75.81	96.54
DyGCN	**79.21**	**97.27**

6 Conclusions and Future work

In this paper, we put forward a dynamic graph convolution network DyGCN for anomaly network traffic detection. We present an effective contrastive learning strategy based on the pattern of normal network traffic. In order to capture the abnormal substructure in the graph, we advance an edge anomaly probability-based anomaly detection method. Experimental results on two datasets demonstrate the effectiveness and efficiency of the proposed method.

In this work, the existing open-source network traffic datasets don't provide attributes of hosts which can be used to improve the node representation learning ability of the model. Therefore, we will further consider how to generate attributes of nodes and incorporate node information efficiently. In addition, there is a certain gap between the real network and the open-source datasets which have a small network scale, simple communication structure, and incomplete attack events. We will consider building a new data set to verify the effectiveness of the model in the future.

References

[1] Wasim A Ali, KN Manasa, Malika Bendechache, Mohammed Fadhel Aljunaid, and P Sandhya. A review of current machine learning approaches for anomaly detection in network traffic. *Journal of Telecommunications and the Digital Economy*, 8(4):64–95, 2020.

[2] Yifan Feng, Weihong Cai, Haoyu Yue, Jianlong Xu, Yan Lin, Jiaxin Chen, and Zijun Hu. An improved x-means and isolation forest based methodology for network traffic anomaly detection. *Plos one*, 17(1):e0263423, 2022.

[3] Chonghua Wang, Hao Zhou, Zhiqiang Hao, Shu Hu, Jun Li, Xueying Zhang, Bo Jiang, and Xuehong Chen. Network traffic analysis over clustering-based collective anomaly detection. *Computer Networks*, 205:108760, 2022.

[4] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial ex-

amples. In *3rd International Conference on Learning Representations, ICLR*, pages 1–10, 2015.

[5] Francesco Zola, Lander Segurola-Gil, Jan Lukas Bruse, Mikel Galar, and Raúl Orduna-Urrutia. Network traffic analysis through node behaviour classification: a graph-based approach with temporal dissection and data-level preprocessing. *Computers & Security*, 115:102632, 2022.

[6] Jun Zhao, Xudong Liu, Qiben Yan, Bo Li, Minglai Shao, and Hao Peng. Multi-attributed heterogeneous graph convolutional network for bot detection. *Information Sciences*, 537:380–393, 2020.

[7] Wenchao Yu, Wei Cheng, Charu C Aggarwal, Haifeng Chen, and Wei Wang. Link prediction with spatial and temporal consistency in dynamic networks. In *IJCAI*, pages 3343–3349, 2017.

[8] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017*, pages 1024–1034, 2017.

[9] Guo JY, Li RH, Zhang Y, and Wang GR. Graph neural network based anomaly detection in dynamic networks. *Journal of Software*, 31(3):748–762, 2020.

[10] Annamalai Narayanan, Mahinthan Chandramohan, Lihui Chen, Yang Liu, and Santhoshkumar Saminathan. subgraph2vec: Learning distributed representations of rooted sub-graphs from large graphs. *CoRR*, abs/1606.08928, 2016.

[11] Wai Weng Lo, Siamak Layeghy, Mohanad Sarhan, Marcus Gallagher, and Marius Portmann. E-graphsage: A graph neural network based intrusion detection system. *CoRR*, abs/2103.16329, 2021.

[12] Siddharth Bhatia, Bryan Hooi, Minji Yoon, Kijung Shin, and Christos Faloutsos. Midas: Microcluster-based detector of anomalies in edge streams. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI*, pages 3242–3249, 2020.

[13] Yepeng Yao, Liya Su, Chen Zhang, Zhigang Lu, and Baoxu Liu. Marrying graph kernel with deep neural network: A case study for network anomaly detection. In *19th International Computational Science Conference, ICCS*, pages 102–115, 2019.

[14] SU Jiang-jun, DONG Yi-hong, YAN Ming-jiang, QIAN Jiang-bo, and XIN Yu. Research progress of anomaly detection for complex networks. *Control and Decision*, 36(6):1293–1310, 2021.

[15] Guanyi Chu, Xiao Wang, Chuan Shi, and Xunqiang Jiang. Cuco: Graph representation with curriculum contrastive learning. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI*, pages 2300–2306, 2021.

[16] Yingtong Dou, Kai Shu, Congying Xia, Philip Yu, and Lichao Sun. User preference-aware fake news detection. pages 2051–2055, 07 2021.

[17] Minji Yoon, Bryan Hooi, Kijung Shin, and Christos Faloutsos. Fast and accurate anomaly detection in dynamic graphs with a two-pronged approach. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 647–657, 2019.

[18] Dhivya Eswaran, Christos Faloutsos, Sudipto Guha, and Nina Mishra. Spotlight: Detecting anomalies in streaming graphs. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1378–1386, 2018.

[19] Lei Cai, Zhengzhang Chen, Chen Luo, Jiaping Gui, Jingchao Ni, Ding Li, and Haifeng Chen. Structural temporal graph neural networks for anomaly detection in dynamic graphs. In *CIKM '21: The 30th ACM International Conference on Information and Knowledge Management, Virtual Event, Queensland*, pages 3747–3756, 2021.

[20] Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. Dysat: Deep neural representation learning on dynamic graphs via self-attention networks. In *WSDM '20: The Thirteenth ACM International Conference on Web Search and Data Mining*, pages 519–527, 2020.

[21] Jingxin Liu, Chang Xu, Chang Yin, Weiqiang Wu, and You Song. K-core based temporal graph convolutional network for dynamic graphs. *CoRR*, abs/2003.09902, 2020.

[22] David W. Matula and Leland L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *J. ACM*, 30(3):417–427, 1983.

[23] Petar Velickovic, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. *ICLR (Poster)*, 2(3):4, 2019.

[24] Yaochen Xie, Zhao Xu, Zhengyang Wang, and Shuiwang Ji. Self-supervised learning of graph neural networks: A unified review. *CoRR*, abs/2102.10757, 2021.

[25] Bijaya Adhikari, Yao Zhang, Naren Ramakrishnan, and B. Aditya Prakash. Sub2vec: Feature learning for subgraphs. In *Advances in Knowledge Discovery and Data Mining - 22nd Pacific-Asia Conference*, pages 170–182, 2018.

[26] Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. graph2vec: Learning distributed representations of graphs. *CoRR*, abs/1707.05005, 2017.

[27] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196, 2014.

[28] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *CoRR*, abs/1810.00826, 2018.

[29] David M. J. Tax and Robert P. W. Duin. Support vector domain description. *Pattern Recognit. Lett.*, 20(11-13):1191–1199, 1999.

[30] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016.

[31] Youngjoo Seo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson. Structured sequence modeling with graph convolutional recurrent networks. In *Neural Information Processing - 25th International Conference, ICONIP*, pages 362–373, 2018.

[32] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao B. Schardl, and Charles E. Leiserson. Evolvegcn: Evolving graph convolutional networks for dynamic

graphs. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI*, pages 5363–5370, 2020.

[33] Enrique F Schisterman, Neil J Perkins, Aiyi Liu, and Howard Bondell. Optimal cut-point and its corresponding youden index to discriminate individuals using pooled blood samples. *Epidemiology*, 16(1):73–81, 2005.

Yonghao Gu received the Ph.D. degree from the Beijing University of Posts and Telecommunications, China, in 2007. He is currently a Lecturer with the Beijing Key Laboratory of Intelligent Telecommunications Software and Multimedia, School of Computer, Beijing University of Posts and Telecommunications. He has published innovative works in high-level conference proceedings and journals. His current research interests include network security and data mining. He serves as a reviewer for top journals and conference program committees.

Xiaoqing Zhang received the B.S. degree from the Zhengzhou University, China, in 2020. She is currently a master degree candidate with the Beijing Key Laboratory of Intelligent Telecommunications Software and Multimedia, School of Computer, Beijing University of Posts and Telecommunications. Her main research interests include network security.

Hao Xu received the B.S. degree from the University of Electronic Science and Technology of China in 2020. He is currently a master degree candidate with the Beijing Key Laboratory of Intelligent Telecommunications Software and Multimedia, School of Computer, Beijing University of Posts and Telecommunications. His main research interests include network security.

Tiejun Wu received the B.S. degree in Computer Science and Technology from Huazhong University of Science and Technology in 2002, and the master's degree in Software Engineering from Huazhong University of Science and Technology in 2006. He is currently working for NSFOCUS. Research interests include APT, botnet and other advanced cyber security threats, and advanced threat hunting.