Dynamic DDoS Attack Detection in Software-Defined Networks Using Graph-Based Machine Learning

1st Maya Kapoor
College of Computing and Informatics
University of North Carolina at Charlotte
Charlotte, NC, USA
mkapoor1@uncc.edu

***Abstract*—The novel architecture of software-defined networking has brought routing optimization to the internet, but has also introduced new attack vectors for distributed denial of service (DDoS) attacks. A resilient SDN must be capable of detecting anomalous behavior in an efficient, automated manner in order to defend against these attacks. Modeled as a provenance graph, the network topology and net flow features may be used to perform graph machine learning analysis for anomaly detection. In order to capture complete dependencies and perform targeted defense, this paper proposes an SDN-DDoS detection system capable of modeling both control and data plane provenance as a multivariate time series of graphs in order to perform edge-level anomaly detection. First, provenance data is gathered from multiple planes of the SDN architecture. We then compare and contrast several spatial and spectral node embedding techniques to learn structural properties and embed these with the edge features. We then perform edge anomaly detection as a binary classification problem and use recorded provenance information to reflexively return anomalous flows at the port-level for targeted defense and mitigation. Our results show a 2-layer GraphSAGE node embedding to be the most informative, and that the multi-layer provenance approach enhances model accuracy and detection capability.**

***Index Terms*—Software-Defined Networking (SDN), Graph Embeddings, Distributed Denial of Service (DDoS), Data Provenance**

I. Introduction

Distributed Denial-of-Service, or DDoS attacks are one of the most detrimental threats to modern communication networks. In this scenario, an attacker attempts to deny service to a node in the network by flooding links, switches, or some device with spoofed packets or requests, thus overwhelming network resources and preventing legitimate traffic from reaching the victim. DDoS attacks have a particularly significant impact on software-defined networks because attacks may be executed at application, data plane, and control plane layers. For example, at the control layer SDN centralizes activity and thereby increases the potential for single points of failure if the controller becomes the victim of a DDoS attack. An attacker may focus on creating a large number of attack flows not matching configured rules, thus overwhelming the southbound interface of the controller with `packet_in` messages. The attacker may also choose to approach via the data plane and invade hosts in the network, turning them into bots which exhaust memory and bandwidth [1]. With the advent of software-defined networking in IoT, smart grid, SCADA, and other such types of computer networks, the potential for DDoS and the need for detection and mitigation are also on the rise.

It is given that during a DDoS attack, the network state will undergo some change, and this will appear as an anomaly in network representations. Thus, anomaly detection algorithms are an apt approach to detecting DDoS attacks. The network nature of this problem also makes it a use case for graph neural networks (GNNs), where data is modeled as nodes and edges and deep learning is used to calculate hidden features of the network. GNNs are capable of learning structural embeddings of nodes, where nodes are characterized by their location in the graph, their own connections, and the traits of their neighbors. In addition to structural features, GNNs may work on attributed graphs and consider statistical data like packet rate, byte count, and flow duration in order to inform the training model. The combination of multiple embeddings along with multiple layers of provenance information from application, control, and data plane layers of SDN should provide a rich context for learning expected behavior and detecting deviant activity.

II. Problem Statement

The SDN is a dynamic graph, meaning that nodes and edges may appear or disappear at any given time in the network. Furthermore, in DDoS attacks the state of a node may change if a legitimate device becomes infected or incorporated into the attacker's botnet; legitimate devices are not immune to becoming corrupted. We assume a node is characterized by the events in which it is involved with its neighbors, an assumption affirmed in related works in this area [1]–[3]. In addition to node classification, the interaction between nodes,

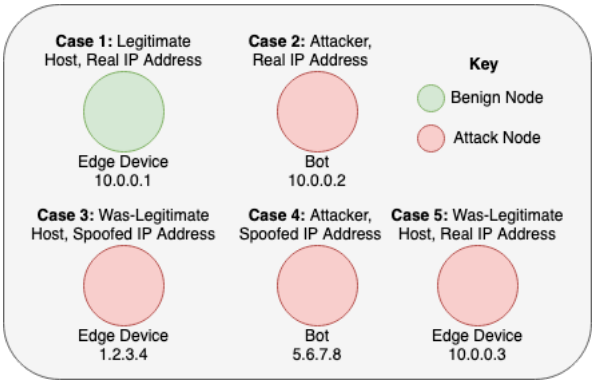

Fig. 1. Binary classification of SDN endpoint devices, or the nodes in the network.

i.e. the traffic flow that forms an edge may also be classified as anomalous or normal.

Let N be a software-defined network with a set of edges E and a set of vertices V. At time t_1, vertex v_2 is not a member of the graph, $v_2 \notin V$, but $v_1 \in V$. At time t_2, the vertex v_2 joins the software-defined network in a default-allow security policy, so $v_2 \in V$. The node classification problem is that both nodes v_1 and v_2 must be one of five categories as shown in Figure 2, but this ultimately becomes a binary classification problem of malicious or benign.

Cases 1, 2, and 4 describe devices joining the network like v_2 in the network. Cases 3 and 5 deal with hosts previously existing in the network like v_1. Case 1 describes a legitimate edge device attempting to communicate with another device in the SDN and thus joining the network. This device would be using a legitimate IP address; note that this may not necessarily correspond to the device itself but is permissible to be behind a private network or VPN. Case 2 and case 4 both describe devices which enter the network infected, likely members of botnets in a DDoS attack. The second case covers an attacker using a real IP address, and the fourth case a spoofed IP address. Cases 3 and 5 similarly cover both address cases, but for hosts which were previously legitimately communicating with the network. In these scenarios, the devices have become infected and it is assumed the traffic pattern will change with an increased number of `packet_in` requests or some type of flooding strategy.

Classifying node endpoints as malicious or benign would allow an administrator to craft a blacklist policy to prevent DDoS attacks. Unfortunately, this problem is more complex if endpoints cannot be trusted to report their identification accurately and can forge IP addresses as seen in cases 3 and 4. Blacklisting this IP address would serve little purpose if the attacker can simply spoof another address and repeat the attack. An alternate approach would be to mitigate the threat at the control plane layer in the software switch which the administrator has domain over. In our threat model the attacker has not infiltrated the control plane and thus has no ability to modify this configuration.

Knowing which switch to adjust requires detecting anomalous flows and that shifts the model focus to edge anomaly detection. In terms of mitigation strategy, altering switch flow can also be disruptive to good network behavior. Shutting down or throttling flows at the switch when DDoS attacks are detected has been proposed in [1], but this also hinders regular network traffic which furthers the attacker's own purpose to reduce network capacity and could prevent mission critical network activity.

Instead of switch-level mitigation, we propose a port-level strategy as a more targeted defense approach [4]. Anomalous switch-port pairs can be identified through the flows passing through them via edge anomaly detection. We leave the node classification problem as an aspect for future work. By combining the information available in a multi-layer system, we can reduce the impact security has on regular network flow while still maintaining effective resilience.

III. Contributions

We provide the following contributions to the state of the art in the intersection of SDN-DDoS detection and graph machine learning:

- A *resilient detection strategy* capable of reporting anomalous switch-port behavior,
- A graph neural network model based on *clique embedding* which accurately classifies anomalous edges seen in the SDN-DDoS dataset,
- A *comparison of the state of the art structural embedding methods* for GNNs on a real SDN-DDoS dataset,
- A *multi-layer provenance model* to provide semantic context to graph representation learning for better results.

IV. Background

A. Software Defined Networking

Traditional networking leaves controlling data flow up to hardware devices like switches and routers. These must be configured with routing information, or the routing information propagated algorithmic-ally through the network and stored in routing tables. A more modern solution which leverages virtualization is software-defined networking. Instead of using hardware devices, SDN emulates a software controller and manages network devices through virtual machines and hypervisors in order to centralize control activity. Traditional hardware continues to propagate the traffic load, but administrators may now apply programming and automation to the control plane instead of configuring narrow-minded devices [5].

A software defined network is made up of several layers. At the application layer there are typical network applications, firewalls, intrusion detection systems, and other endpoint or user-level software. The control plane is where the SDN

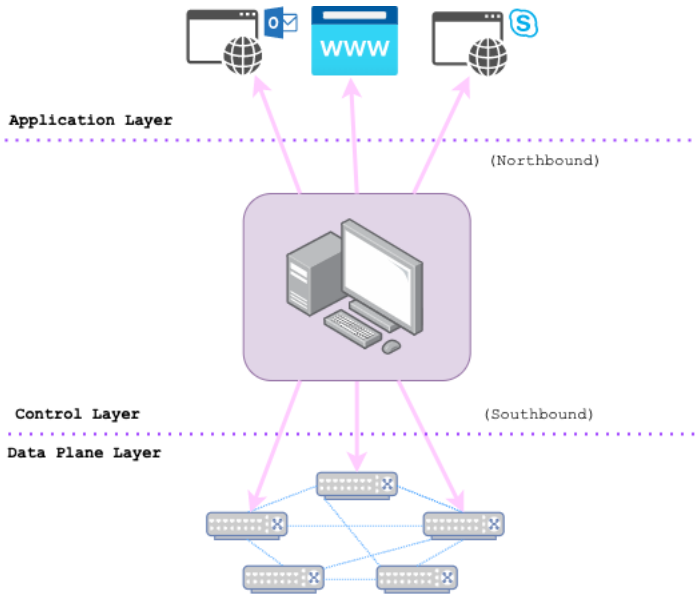

Fig. 2. A simple SDN network with three architectural layers.

controller software resides along with network configurations and flow tables. Lastly, the infrastructure or data plane layer is the physical hardware and switches which propagate the traffic. The control layer communicates with the application layer through the northbound interface and with the data plane layer through the southbound interface [4].

B. Distributed Denial of Service Attacks in SDN

While this high-level control in SDN is more efficient than a hardware-based packet engine, the centralization of network decision-making creates a risk of single points of failure for the system security. The transition to a software platform also increases vulnerabilities. Furthermore, the various layers in SDN architecture each expose themselves to unique versions of DDoS attacks [5]:

- **Application Layer:** An attack may occur on a particular application which exhausts resources and causes system delay or service interruption. An example of this would be flooding a web server hosting `google.com` with HTTP requests so that legitimate users could not access the page.
- **Control Layer Buffer:** If the attacker sends in a large number of `packet_in` requests to the controller, the controller may be forced to drop benign requests from its overfilled buffer.
- **Control Layer Link:** the `packet_in` requests may also flood the link between data plane and control layers, denying legitimate service.
- **Data Plane Layer:** the attacker may spoof IP addresses when sending multiple `packet_in` requests, and these may overfill the flow tables and thus deny slots to legitimate users.

C. Data Provenance

Described as the lineage of data, provenance tells us where data originated, who and what modified the data, and what emanated from it. Importantly, provenance models record data over time and thus can track temporal relations and attacks over time. Data gathered from the application, control, and data plane layers may be treated as provenance and modeled accordingly to detect attacks and aide in root cause analysis when anomalies are found. Authors of PicoSDN [4] emphasized the importance of a multi-layer provenance system particularly in SDN as single-layer provenance does not provide enough semantic context to capture all attacks. In our mitigation strategy, we require port information as well as switch information which spans multiple layers of the network provenance framework. Our graph embedding features are also gathered across provenance layers and combined in the learning model for maximized context of network activity.

D. Graph Neural Networks

Convolutional neural networks have been applied in data sets which may be represented as sequences or grids, for example pixels in images. Input data is passed through convolutional filters which learn additional hidden features about it. The fixed structure of the data is a limitation for CNNs' applicability. Networks are useful for modeling more complex data which cannot be represented in such a fixed manner, for example relationships between individuals, protein sequences, and computer system provenance. Graph neural networks combine the advantage of knowledge gleaned through structural properties of graphs and machine learning on node and edge attributes. They accomplish this through graph embedding, which is the process of converting high dimensional graph properties to lower dimensional representations such as a feature-vector, hash, or eigenvalue.

There have been many approaches proposed as embedding layers to produce the most informative embeddings, including Euclidean-based distance measures [6], [7], spectral projections [8], random walks [9], and self-attention mechanisms [10]. The more popular methods can be divided into spatial, spectral, and attention-based categories as well as intersections of these for a multi-embedding. The approach to embedding nodes and edges is a design feature which can dramatically affect the outcome of the machine learning model for classification. Choosing the best representation is a matter of both mathematical soundness and the nature of the data set. In essence, a node embedding characterizes a node by the characteristics of its neighbors. This function is known as neighbor aggregation. The expressive power of a graph neural network model is determined by the neighbor aggregation function it utilizes. In our model, we compare four approaches to embedding. Two of the methods, GraphSAGE [6] and GCN [7], are based on spatial properties. For a spectral angle, we use Deep Graph Library's Chebyshev [8] convolution layer. Lastly, we adapt the clique embedding inspired by the

skip-gram architecture with random walks that is used in NetWalk [9] as an embedding for our model.

1) ***GCN****:* Let $\mathcal{N}(i)$ be the set of neighbors of node *i* and c_{ji} be the product of the square root of node degrees, $\sqrt{|\mathcal{N}(j)|} \times \sqrt{|\mathcal{N}(i)|}$. Also, let σ be the activation function. GCN [7] is mathematically described by this equation:

$$h_i^{(l+1)} = \sigma(b^{(l)} + \sum_{j \in \mathcal{N}(i)} \frac{1}{c_{ji}} h_j^{(l)} W^{(l)}) \tag{1}$$

This uses element-wise mean pooling over the features of its neighboring nodes. It is then followed by a non-linear activation function such as ReLU, and repeated for the number of desired layers.

2) ***GraphSAGE****:* The original paper for this model describes itself as an inductive transformation of nodes based on its neighbors. Let $\mathcal{N}(i)$ be the set of neighbors of node *i*. The algorithm can be described as shown here:

$$\begin{aligned} h_{\mathcal{N}(i)}^{(l+1)} &= aggregate(\{h_j^l, \forall j \in \mathcal{N}(i)\}) \\ h_i^{(l+1)} &= \sigma(W \cdot concat(h_i^l, h_{\mathcal{N}(i)}^{l+1}) \\ h_i^{(l+1)} &= norm(h_i^l) \end{aligned} \tag{2}$$

This method uses element-wise max pooling over the features of its neighboring nodes to sample and aggregate (SAGE), and then also follows it with a non-linear activation function repeated for the number of desired layers. The crux of this algorithm is that a node may be summarized by its neighbors, and thus the embedding is induced through neighborhood aggregation [6].

3) ***ChebConv****:* In contrast to the aforementioned spatial techniques, the spectral technique described in [8] uses a Fourier Transform that is optimized with Chebyshev expansion for computational efficiency. Let $\tilde{\mathrm{A}}$ be $A+I$, W and a learnable weight and $\tilde{\mathrm{L}}$ the graph laplacian matrix, then we can describe the Chebyshev convolutional layer as follows:

$$\begin{aligned} h_i^{(l+1)} &= \sum_{k=0}^{K=1} W^{k,l} z_i^{k,l} \\ Z^{0,l} &= H^l \\ Z^{1,l} &= \tilde{\mathrm{L}} \cdot H^l \\ Z^{k,l} &= 2 \cdot \tilde{\mathrm{L}} \cdot Z^{k-1,l} - Z^{k-2,l} \\ \tilde{\mathrm{L}} &= 2(I - \tilde{\mathrm{D}}^{-1/2} \tilde{\mathrm{A}} \tilde{\mathrm{D}}^{-1/2})/\lambda_{max} - I \end{aligned} \tag{3}$$

4) ***NetWalk****:* NetWalk [9] is a proposed algorithm for embedding dynamic graphical data into lower-dimensional feature vectors for vertices and edges. In their original publication, NetWalk authors injected anomalies into a streaming graph and performed anomaly detection on nodes and edges with better performance than node2vec, DeepWalk, spectral clustering, and several other strategies.

The first step to NetWalk is generating the random walks. The formal definition is provided here:

Definition 1 (Network Walk). For a given vertex $v_1 \in V$ in a network $G(E, V)$, its network walk set is defined as $\Omega_{v_1} = \{(v_1, v_2, ...v_l) | (v_i, v_{i+1}) \in E \wedge p(v_i, v_{i+1}) = \frac{1}{D_{v_i, v_i}}\}$, which is a collection of *l*-hop walks starting from vertex v_1. The transition probability $p(v_i, v_{i+1})$ from v_i to v_{i+1} is proportional to the degree D_{v_i,v_i} of vertex v_i. We call Ω_v a network walk set starting from v and $\Omega = \{\Omega_v\}_{v \in V}$ as the union of all walks [9].

After the walks are generated, node representations are learned following a regression/optimization problem. Their *clique embedding* uses a feed-forward auto encoder to learn the vector representation and minimize pairwise distances among all vertices in each walk. The inputs and outputs are one-hot encoded vectors, and the goal is to learn a latent representation for each network walk. Much like skip-gram, the goal is to minimize the cost function $J(W, b)$ as a function of W and b by computing the partial derivatives of the equation [9]:

$$\begin{aligned} J(W,b) = & \underbrace{\sum_{i=1}^{|\Omega|} \sum_{1 \le p,q \le 1} \left\| f^{(\frac{n_l}{2})}(x_p^{(i)} - f^{(\frac{n_l}{2})}(x_q^{(i)} \right\|_2^2}_{\text{Clique Embedding Loss}} \\ & + \underbrace{\frac{\gamma}{2} \sum_{i=1}^{|\Omega|} \sum_{p=1}^{l} \left\| f^{(n_l)}(x_p^{(i)}) - x_p^{(i)} \right\|_2^2}_{\text{Reconstruction Error}} \\ & + \underbrace{\beta \sum_{\ell=1}^{n_l - 1} \sum_j KL(\rho || \hat{\rho}_j^{(\ell)})}_{\text{Sparsity Constraint}} + \underbrace{\frac{\lambda}{2} \sum_{\ell=1}^{n_l} \left\| W^\ell \right\|_{F'}^2}_{\text{Weight Decay}} \end{aligned} \tag{4}$$

In the formula, $\{x_p^{(i)}\}_{p=1}^l$ represents the network walk, l is the walk length, $n_l > 2$, $f^{(\frac{n_l}{2})}(x_p^{(i)})$ is the linear transformation of weight matrices (W) and bias vectors (b) with sigmoid activation, and ℓ_2 normalization is used to minimize reconstruction error. A sparsity parameter ρ is applied with the Kullback-Leibler divergence. The weight decay term helps prevent overfitting by decreasing the magnitude of the weights along with penalty terms γ, β, and λ. The authors of NetWalk provide an algorithmic analysis in their publication, a Laplacian-matrix form of the final equation, and an example of error terms computed for the back-propagation step following the encoder before partial derivatives are taken. Then, the derivative result is used to determine the appropriate step to take toward optimization [9].

V. Datasets

Across the DDoS-SDN research space, one common deficit is the lack of publicly available datasets which contain DDoS on real software-defined networks. Many proposed detection strategies conducted their experiments on non-SDN data sets like ISCX, NSL-KDD, and KDD-cup99 [5], or the datasets are not publicized [3]. Ahuja et al. provide a simulated DDoS attack dataset on an SDN network that is publicly available and described in detail in their system paper [5]. This dataset was generated using the Mininet emulator and RYU controller and contains provenance information from application, data, and control plane layers in the network. Three different network topologies were emulated which are shown in Figure 3. The configuration of the simulation environment is given in detail in Table II. There are several host nodes which will produce malicious DDoS traffic toward the victim machine in the network. The design is such that malicious host `H9` is isolated to its own switch, but for example in the first topology `H1` and `H11` are connected to switches with benign traffic. Similar setups may be seen in the other network designs. For this reason, controlling the overall switch flow will harm benign traffic flow on these nodes. This shows the importance of switch-port isolation and mitigation in our model, to avoid interfering with `H2`'s and `H12`'s experience.

In this network, a DDoS attack was determined to occur if traffic was seen that violated one of the following four constraints [5]:

1) $0 \leq P < \Theta$. An attack is said to take place if the packet count P exceeds some threshold Θ.
2) $0 \leq P_{in} < \beta$. An attack is said to occur if the `packet_in` count is not greater than zero or exceeds some threshold β.
3) $D \in 0, 1$. The decision variable D must be a binary value 0,1 which corresponds to benign and malicious traffic, respectively.
4) $\sum_{i=1}^{n} F \leq \zeta$. The final constraint is that the number of flow entries (F) present in the switch should not be more than the processing capacity of the emulator (ζ).

These constraints map to the identified attack vectors across the multiple planes in the SDN architecture. The dataset contains information written to CSV from the RYU controller with 104,345 records with 23 features each. Benign traffic was generated using the `mgen` traffic generation tool at a rate of 450 packets per second. The malicious traffic was generated using spoofed IP addresses and the `hping3` tool at the same rate. Attack traffic generated was comprised of TCP-SYN attacks, UDP flood attacks, and ICMP flood attacks, and benign traffic consisted of TCP, UDP, and ICMP traffic [11].

Ahuja et al examine and compare results on this dataset using classical machine learning classifiers like Random Forest, K-Nearest Neighbors, Ensemble Methods, SVC, and Logistic Regression. We extend this to graph neural network model analysis using various embedding methods and comparing results [5].

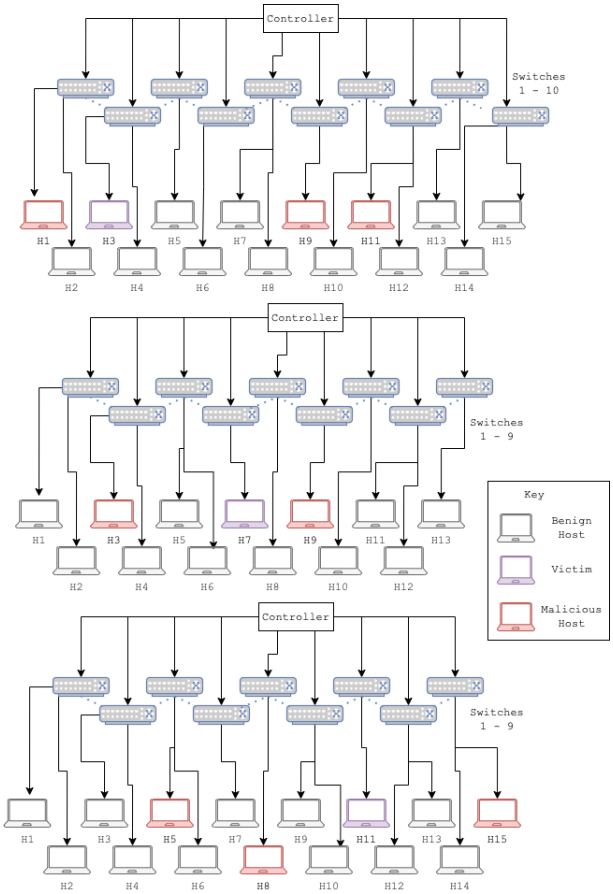

Fig. 3. Topologies for SDN-DDoS attack dataset.

TABLE I
Traffic Class Instances (Num. Packets)

Traffic Class	Benign	Malicious	**Total**
ICMP	24957	16364	**41321**
UDP	22772	10816	**33588**
TCP	18897	10539	**29436**
Total	**63561**	**40784**	

In our experiments, the dataset was pre-processed to normalize edge features between [0,1] for statistical analysis. Furthermore, we translated the `src_IP` and `dst_IP` fields into indexed nodes with a single row of the dataset representing an edge in the graph between the nodes. In our model, we wanted to train only on benign data and then detect attacks as unseen anomalies. A detection mechanism optimized to this attack vector is particularly important for zero-day exploits where the system may be able to detect and respond to previously unforeseen attack patterns. This approach differs from the original use of the dataset and thus is a novel approach to the problem in this network. In order to accomplish this, we created a training set by splitting the data into attack and benign, then dividing a portion of the benign for the

TABLE II
SIMULATION ENVIRONMENT PARAMETERS

Parameters
Host OS: Windows
Guest OS: Ubuntu 16.04
Hypervisor: VirtualBox 5.1.26
Emulator: Mininet
Controller: Ryu
Number of Controllers: 1
Number of Switches: 9-10
Number of Hosts: 13-15
Protocol Used: OpenFlow
Graphical Package: MiniEdit
Traffic Generation Tool: mgen, hping
Controller Port Number: 6653
Simulation Time: 250 minutes
Statistics Collection Interval: 30s
Bandwidth Plot Interval: 30s
Number of topologies: 3

training and testing sets. This gave us a final result of 47,670 benign training edges and 40,784 attack testing edges plus 15,891 benign testing edges. We still preserve the date/time stamp ordering to include temporal properties of the data. Training was conducted on a static graph representation of the benign edges, and testing done in time-stamp sliced batches to simulate real-time detection on a streaming graph.

We use both node embeddings and edge features for classification. The flow data proved to be powerful not only in our experiments but across previous research works [1], [3], [12] and in the original tests on this dataset [5]. We use the same features they calculated:

- $F = length(flowtable)$*:* The number of flow entries is important for classification because periodic monitoring of the switch should reveal target switches which have reached the maximum number of flow table entries and thus may be subject to attack by a malicious host on a port.
- $APPF = \sum_{i=1}^{i=n} c_i/F$*:* The average packet count per flow is a significant statistic because it tracks an increase in packet count without relying on a potentially spoofed IP address for identification.
- $APBF = \sum_{i=1}^{i=n} b_i/F$*:* This calculation is similar to average packet count per flow but uses the average byte count instead. It may also be indicative of a flooding attack.
- $Total\ duration = ((d \cdot 10^9) + d\_nano)$*):* In general, attack traffic lasts for a longer duration than benign traffic []. Thus, we can use the total duration of flows from the dataset as an indicator for attack.
- $Packet\ rate = p_{t+1} - p_t/i$*:* When an attacker uses spoofed IP addresses, the controller receives a large number of `packet_in` messages which are one method of performing DDoS on the SDN controller. For this reason, the *packet_in* rate can be indicative of an attack. In the formula, t and $t+1$ represent fixed times, while i is the interval monitoring rate where the SDN controller records the number of `packet_in` events.
- $Bandwidth = (t \cdot 8)/1000 + (r \cdot 8)/1000$: A decrease in port bandwidth can also be indicative of an attack as the malicious host will send more control messages and requests and less data. This value can be calculated from `tx_bytes` and `rx_bytes` extracted from the port statistics.

These features are representative of multiple layers from the SDN architecture. For example, the recorded switch IDs and port IDs are provenance from the data plane. The flow statistics are from the controller. The protocol type which is also recorded come from the application layer and the app configurations themselves. Getting the full context of the network from multi-layer provenance improves the accuracy of the machine learning models in both our experiments and previous work in this dataset.

VI. METHODOLOGY

Neural networks require fine tuning of hyper parameters in order to achieve optimal performance. Furthermore, selection of embedding techniques and features dramatically effects performance as well. Our model supports the embedding methods discussed in the background section and may be tuned with a stochastic or momentum based optimizer for future adaptability to new data sets. We use the Deep Graph Library [13] built on PyTorch [14] in order to construct the neural network model and model the graph internally. For the NetWalk embedding, we modify the original NetWalk source code which is written using TensorFlow [15] to adapt it to be used in DGL. The source code for our implementation can be found on GitHub[1].

The first step for the model is to learn the hidden features and node representations. We approach this by first creating a one-hot tensor out of the node adjacency list from the graphs. We then feed this into our chosen convolutional layer. The details of these layers are given in the background section. In between convolutional layers we apply a ReLU activation function.

In order to combine the node features with edge features for edge anomaly detection, we encode the representation of source and destination nodes using the Hadamard operator and concatenate this result with the calculated statistical edge features. The Hadamard product of two matrices is given as $(A \circ B)_{ij} = (A \circledcirc B)_{ij} = (A)_{ij}(B)_{ij}$ for matrices of the same dimension (otherwise it is undefined) [9]. The encoding method is flexible; our model can also be configured to encode based on an average, weighted $\ell 1$, or weighted $\ell 2$ normalization.

The final layer of the model applies the statistical edge features to the graph as well as the encoded, structural features which characterize the nodes. We then feed this input into

[1]https://github.com/mayakapoor/ProvSDN

two linear transformation layers with ReLU activation between them and a final sigmoid layer for binary classification. The flow of the model from input to final result is shown in Figure 4.

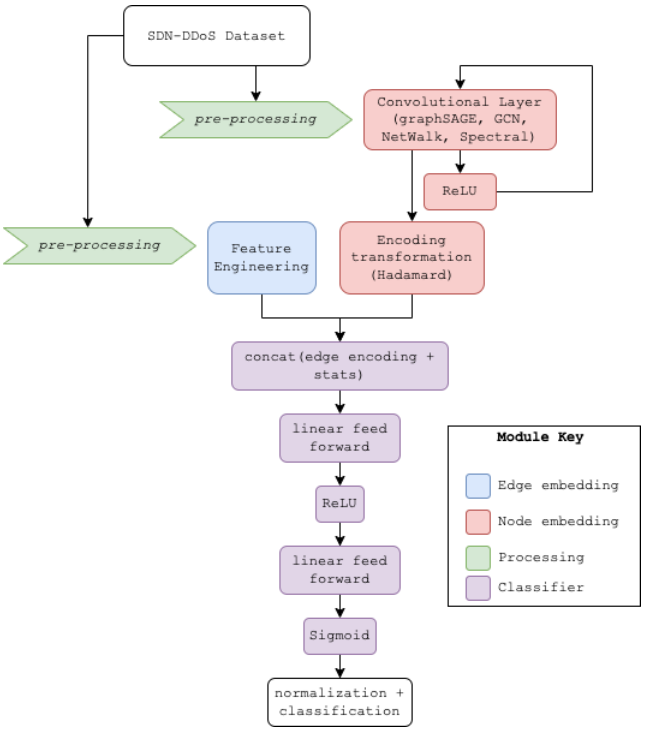

Fig. 4. Hierarchical view of the GNN model flow for SDN-DDoS attack detection.

We trained our model on the benign attack data and optimized with the Adam optimizer and binary cross entropy as the loss function. We also experimented with various configurations in order to determine what combinations were most effective in detecting attacks. In order to test the capability of structural embedding for the detection task, we isolated the node embedding features and performed tests using only that version of the model with additional variations. Specifically, we adjusted the following:

1) Number of convolutional layers (1-3),
2) Node embedding method, either spatial (GCN, GraphSAGE, NetWalk) or spectral (Chebyshev Filter),
3) and encoding method to form edges (Hadamard, Average, Weighted L1, or Weighted L2).

We additionally tuned the optimizer used for training, but did not find the results to be significantly different enough to include here. Following training, we simulated a streaming detection scenario by dividing the testing edges into graph slices with 100 edges in each slice. The slices were fed in an iterative manner into the model which reported its findings and returns the ID of edges which are reported to be anomalous. This can be mapped to the stored dataset which contains the switch/port pair for targeted defense and mitigation of the offending flow.

For evaluation metrics, we use the area underneath the ROC curve which indicates a binary classifier's ability to distinguish between the classes. Additionally, we report values derived from the confusion matrix of the results. This table is used to describe the performance of classification models and is also especially prevalent in cybersecurity (shown in Figure 5).

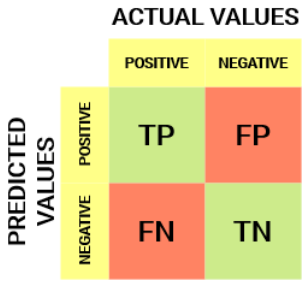

Fig. 5. A generic confusion matrix showing prediction relations to ground truth.

We provide an explanation of the metrics as they relate to our problem statement:

- **True Positive (TP)**: Classified DDoS, was DDoS.
- **False Positive (FP):** Classified DDoS, was normal.
- **False Negative (FN):** Classified normal, was DDoS.
- **True Negative (TN):** Classified normal, was normal.

- **Accuracy:** The percent the classifier predicted DDoS or normal correctly. $= \frac{(TP+TN)}{total}$
- **Precision:** The percentage of DDoS predictions that are correct. $= \frac{TP}{TP+FP}$
- **Recall:** Out of all the DDoS traffic, the percentage correctly identified. $= \frac{TP}{TP+FN}$
- **F1 Score:** harmonic mean of precision and recall. $= \frac{2 \cdot (P \cdot R)}{(P+R)}$

- **True Positive Rate:** When the traffic is DDoS, this indicates how often the prediction is correctly DDoS. $= \frac{TP}{Actual\ DDoS}$
- **False Positive Rate:** When the traffic is normal, this indicates how often the prediction is wrongly DDoS. $= \frac{FP}{Normal\ Traffic}$
- **AUC ROC:** A plot of the True Positive rate along the y-axis and the False Positive rate along the x-axis. The area under the ROC curve indicates the classifier's ability to distinguish between normal and DDoS traffic.

VII. Results

The first research question we wanted to formally explore was the ideal node embedding which would ultimately combine with edge features. Before adding edge features to our model, we ran training simulations and tests on the DDoS-SDN dataset [11] using only the structural embedding as input features. This feature isolation allows comparison of only the node structural embedding in order to determine which was the most informative for the dataset.

For training the models, we ran a 20-epoch loop and used Adam as the optimizer with a learning rate = 0.01. All training data were benign, routine SDN network flow graphs. The testing data was a mixture of DDoS and benign flows as described in the datasets section and edges were classified as either malicious or benign. For the GraphSAGE, GCN, and Chebyshev models we tested models with 1-layer, 2-layer, and 3-layer configurations. We also tested NetWalk's performance on the dataset. For evaluation metrics, we chose to compare accuracy, precision, recall, F1-score, and AUC-ROC measures. Results are shown in Table III. Overall, initial results from feature isolation indicated that of the models tested, the 2-layer GraphSAGE embedding was most capable of deriving a structural, discriminating characterization of routine network behavior. We chose to adjust layers in order to assess how expansion of the neighborhood would affect model accuracy; in general, we found that two layers were sufficiently expressive. In the case of GCN, one layer achieved the best results.

TABLE III
NODE EMBEDDING FEATURE ISOLATION RESULTS

	Accuracy	Precision	Recall	F1 Score	AUC ROC
GCN					
1-Layer	67.90%	77.89%	77.40%	77.64%	0.605259
2-Layer	67.79%	77.77%	77.40%	77.59%	0.603201
3-Layer	67.23%	77.33%	77.33%	77.10%	0.595724
GraphSAGE					
1-Layer	61.42%	73.76%	72.05%	72.89%	0.531375
2-Layer	**68.64%**	**78.33%**	**78.04%**	**78.19%**	**0.613337**
3-Layer	53.46%	67.69%	67.65%	67.67%	0.423799
Spectral Filtering					
1-Layer	56.52%	70.92%	67.12%	68.97%	0.482711
2-Layer	60.78%	72.78%	72.72%	72.75%	0.514869
3-Layer	55.80%	69.33%	69.23%	69.28%	0.453373
NetWalk	59.96%	72.43%	71.68%	72.05%	0.508464

Secondly, we wanted to determine the effects of the embedding encoding method on performance. In order to do this, we ran the 2-layer version of each model with each encoding method from the NetWalk publication with the same hyper parameters as the feature isolation test. Results are shown in Table IV.

Our final round was to run the entire model with both edge features and the node representation embeddings to determine the most effective classifier from our model set. With accurate classification, the model may report its findings and in the context of an embedded resilient system, return a matching index to the switch/port pair for the anomalous flow detected. We ran our model on training data from two scenarios: first, we trained on both benign and attack data using sci-kit tools for splitting datasets. This was to simulate detecting attacks which had previously been seen. In the second scenario, we trained only on benign data and tested on attack and benign edges in order to simulate a zero-day attack scenario where the attack model has not been previously seen. We also increased the epochs to 50, but still used Adam optimizer with the same learning rate. Table V and Table VI show the results.

TABLE IV
ENCODING METHOD VARIATION RESULTS

	Accuracy	Precision	Recall	F1 Score	AUC ROC
Hadamard					
GCN	67.99%	77.77%	77.40%	77.59%	0.603201
GraphSAGE	**68.64%**	**78.33%**	**78.04%**	**78.19%**	**0.613337**
Spectral Filtering	56.52%	70.92%	67.12%	68.97%	0.482711
NetWalk	59.96%	72.43%	71.68%	72.05%	0.508464
Average					
GCN	67.77%	77.74%	77.40%	77.57%	0.602758
GraphSAGE	55.98%	69.91%	68.21%	69.05%	0.464645
Spectral Filtering					
NetWalk	58.83%	71.74%	70.63%	71.18%	0.496427
Weighted L1					
GCN	48.92%	65.00%	62.90%	63.93%	0.380351
GraphSAGE	55.79%	70.11%	67.26%	68.66%	0.468560
Spectral Filtering	56.78%	70.28%	69.25%	69.76%	0.470592
NetWalk	61.18%	73.67%	71.70%	72.67%	0.529947
Weighted L2					
GCN	61.18%	73.67%	71.70%	72.67%	0.529948
GraphSAGE					
Spectral Filtering	55.94%	69.51%	69.10%	69.30%	0.457031
NetWalk	59.59%	72.42%	70.88%	71.64%	0.507937

The most successful combined model overall was the 1-layer GraphSAGE model which achieved 72.52% accuracy on the data set with known attacks. As expected, the zero-day scenario was less accurate, but still showed some distinguishing capability according to the AUC-ROC score. For both scenarios, we return the anomalous switch-port pairs as detected by the model. In their experiments, Ahuja et al [5] achieved much higher accuracy and F1-score from the edge features alone in this data set; in future work, we will aim to better tune parameters and re-evaluate our model to improve.

TABLE V
MODEL DETECTION RESULTS IN ZERO-DAY SCENARIO

	Accuracy	Precision	Recall	F1 Score	AUC ROC
GCN					
1-Layer	64.12%	75.08%	75.07%	75.07%	0.556320
2-Layer	65.63%	76.12%	76.12%	76.12%	0.574959
3-Layer	63.88%	74.91%	74.91%	74.91%	0.553345
GraphSAGE					
1-Layer	65.25%	75.86%	75.86%	75.86%	0.570234
2-Layer	66.07%	76.43%	76.42%	76.43%	0.580436
3-Layer	66.53%	76.75%	76.75%	76.75%	0.586167
Spectral Filtering					
1-Layer	63.80%	74.85%	74.85%	74.85%	0.552339
2-Layer	64.44%	75.30%	75.29%	75.30%	0.560258
3-Layer	64.16%	75.10%	75.10%	75.10%	0.556714
NetWalk	60.19%	72.34%	72.34%	72.34%	0.507609

VIII. RELATED WORK

Detection of DDoS attacks in SDN networks is a major threat and thus a well-studied problem in network security. There are many works related to calculating node-level entropy, classifying via Support Vector Machines and other machine-learning methods, and trend forecasting. We focus on related works in graph machine learning and graph embedding for anomaly detection as well as multi-layer provenance in SDN.

TABLE VI
MODEL DETECTION RESULTS WITH KNOWN ATTACKS

	Accuracy	Precision	Recall	F1 Score	AUC ROC
GCN					
1-Layer	61.61%	50.83%	50.82%	63.54%	0.596912
2-Layer	61.42%	50.43%	50.41%	50.82%	0.594442
3-Layer	60.86%	50.25%	50.24%	50.42%	0.590123
GraphSAGE					
1-Layer	72.52%	64.65%	64.64%	64.65%	0.711092
2-Layer	68.78%	60.08%	60.06%	60.07%	0.672456
3-Layer	71.58%	63.54%	63.54%	63.54%	0.701538
Spectral Filtering					
1-Layer	61.95%	51.28%	51.28%	51.28%	0.600562
2-Layer	61.09%	50.51%	50.51%	50.51%	0.592493
3-Layer	61.54%	50.80%	50.79%	50.80%	0.596339
NetWalk	62.06%	51.21%	51.21%	51.21%	0.601080

A. Structural-Temporal Graph Learning

Graph representation learning has been applied in a variety of communication network routing tasks. GLASS [3] authors were able to use a graph convolutional network and spectral clustering to achieve 84% throughput on an SDN under DDoS attack. Outside of SDN but in a similar realm of IoT, Protogerou et al [16] employed multi-layer perceptrons on a multi-agent system in order to detect anomalies and perform distributed defense. They also introduce the idea of representing a node's vector as aggregated and transformed feature vectors of its neighbors.

One graph learning approach which has been taken is a spatial, temporal graph convolutional network, or ST-GCN [1]. Convolutional neural networks require fewer training parameters, train faster, and are less prone to overfitting than recurrent neural networks while still providing equal learning ability [17]. The further advantages of this model are that considering both space and time disallows an attacker to overcome statistics-based anomaly detection through changing traffic flow patterns. Furthermore, full network graph analysis allows detection systems to find the attack path in the network which enables a distributed defense response.

A caveat to this approach's experimental setup is that they require In-band Network Telemetry (INT) data sampled from the network, which is available only in data plane-programmable SDNs. In our system, data which is more easily gathered from the control plane can be combined with data plane information for a multi-layer provenance approach.

B. Provenance Graphs for SDNs

In gathering provenance for software-defined networks, viewing only control plane data creates an *incomplete dependency* problem [4]. Control plane level topology is also prone to dependency explosion, where events appear related which are actually not due to provenance granularity being too coarse. Lastly, several vulnerabilities have been found to exist which would allow attackers to corrupt the control plane data, thus rendering anomaly detection ineffective or erroneous. In order to accomplish data plane and application layer provenance analysis, solutions like PicoSDN have been proposed to capture event partitioned provenance data for causal graphs [4] in an ONOS SDN framework. This provides multi-layer provenance information which prevents poisoning at solely the control or data plane layer and provides a fuller scope of the network events.

Specifically, PicoSDN is able to identify traffic at the switch port (i.e. host) and not just switch level. This connects provenance gathered from both the control and data plane to solve the incomplete dependency problem. They further use execution partitioning to resolve dependency explosion for a balance between fine-grained provenance and full-scope identification of the anomaly.

We cite PicoSDN as a related work because it captures the effectiveness of multi-layer provenance which we also demonstrate in our system. The dataset which we use to generate results was produced also from multiple layers of network system provenance in practice, but that was not the focus of the authors as it is in PicoSDN. Still, we use this data as an example of multi-layer provenance and leverage the full-scope context it provides to improve mitigation strategy and learning in our model.

C. Node Embedding and Edge Anomaly Detection

One research problem in the SDN graph learning space is that characterization of a node or endpoint must be performed without trusting that endpoint. Furthermore, no prior knowledge of acceptable endpoints should be required, i.e. for white-listing. New nodes should be able to dynamically join the network without previous knowledge or manual configuration. One way of achieving zero-trust here is through analyzing the node's context in the network. It is much more difficult for an attacker to manipulate all the nodes and edges in a network neighborhood than a single node. In order to apply mathematical models to these complex networks, features must be reduced to numerical representations. In recent literature, there have been both probabilistic and embedding-based approaches to anomaly detection of vertices and edges.

Bars and Kalogeratos [2] used clique stream analysis and structural features of the nodes to create "event fingerprints" out of time slices of the graph. Event fingerprints are recreated versions of graphs that are undirected edges without notion of senders and receivers for a given time interval. They then use the sum of Bernoulli conditional probability distributions and are able to derive confidence levels which they compare to a threshold for anomaly detection. In sum, this determines the difference in expected volume of traffic and node behavior and actual observed flow. It is from this work that we borrow the assumption that the multivariate time series of the total number of events (this work's "fingerprints") occurring in the network will characterize the node.

Examples of embedding algorithms used previously in

DDoS-based research include node2vec, spectral clustering, and DeepWalk, all of which similarly create Euclidean representations of the nodes. The anomaly detection systems StreamSpot [12], SpotLight [18], and SnapSketch [19] incorporate embeddings and further apply locality sensitive hashing to time slices of these graph embedding vectors in order to create fixed-size representations of graphs which preserve similarity measures. The generated sketch vectors of these graphs can then be used as input to machine learning algorithms. In their experiments, SnapSketch researchers showed the effectiveness of these techniques in finding DoS attacks in a network. While these works have shown results in DoS attacks, the scope of graph-level anomaly detection is not targeted enough to be useful in creating a resilient adaptation strategy for our purposes.

IX. Conclusion

We propose a graph convolutional network-based system capable of detecting anomalous DDoS attack flows from multiple layers of provenance data gathered from software-defined networks and reporting the switch/port pairs seen for resilient adaptation. Our model learns a representation of the network using GraphSAGE node embedding and combines this with the provenance features for a contextual understanding of network routine. A future phase for this work would be the incorporation of line graphs into the framework to support community detection and clustering of nodes. This would allow for node anomaly detection as deviation from the clusters which is similar in theory to the k-means clustering conducted by NetWalk for their node anomaly detection.

This work improves upon existing methods by proposing a more targeted defense strategy leveraging information gathered from multi-layer provenance. Furthermore, we demonstrate the effectiveness of different types of structural embeddings on this type of software-defined networking graph. We also take a unique perspective on this dataset by attempting to learn from only benign traffic and then classify both traffic types. Our findings indicate the 2-layer GraphSAGE embedding to be most effective, with the edge/flow features providing significant enhancement to the detection model. Even in our model which treats nodes as "featureless," properties may be extracted from even the geometric aspects of a nodes' neighbors. In line with previous researchers' findings, we note that the statistical edge features can play a vital role in detection, as well. Understanding the importance of network science to this problem and uncovering hidden representations will continue to enhance network security practitioners' ability to defend against these DDoS attacks.

References

[1] Y. Cao, H. Jiang, Y. Deng, J. Wu, P. Zhou, and W. Luo, "Detecting and mitigating ddos attacks in sdn using spatial-temporal graph convolutional network," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–1, 2021.

[2] B. L. Bars and A. Kalogeratos, "A probabilistic framework to node-level anomaly detection in communication networks," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019, pp. 2188–2196.

[3] K. Nagaraj, A. Starke, and J. McNair, "Glass: A graph learning approach for software defined network based smart grid ddos security," pp. 1–6, 2021.

[4] B. E. Ujcich, S. Jero, R. Skowyra, A. Bates, W. H. Sanders, and H. Okhravi, "Causal analysis for software-defined networking attacks," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 3183–3200. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/ujcich

[5] N. Ahuja, G. Singal, D. Mukhopadhyay, and N. Kumar, "Automated ddos attack detection in software defined networking," in *Journal of Network and Computer Applications*, vol. 187, 2021. [Online]. Available: https://doi.org/10.1016/j.jnca.2021.103108

[6] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," 2018.

[7] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2017.

[8] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," 2017.

[9] W. Yu, W. Cheng, C. C. Aggarwal, K. Zhang, H. Chen, and W. Wang, "Netwalk: A flexible deep embedding approach for anomaly detection in dynamic networks." New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3219819.3220024

[10] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," 2018.

[11] N. Ahuja, G. Singal, and D. Mukhopadhyay, "Ddos attack sdn dataset," 2020.

[12] E. Manzoor, S. Milajerdi, V. Venkatakrishnan, and L. Akoglu, "Fast memory-efficient anomaly detection in streaming heterogeneous graphs," February 2016.

[13] M. Wang, D. Zheng, Z. Ye, Q. Gan, M. Li, X. Song, J. Zhou, C. Ma, L. Yu, Y. Gai, T. Xiao, T. He, G. Karypis, J. Li, and Z. Zhang, "Deep graph library: A graph-centric, highly-performant package for graph neural networks," 2020.

[14] "Pytorch: An imperative style, high-performance deep learning library," 2019. [Online]. Available: http://arxiv.org/abs/1912.01703

[15] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[16] A. Protogerou, S. Papadopoulos, A. Drosou, D. Tzovaras, and I. Refanidis, "A graph neural network method for distributed anomaly detection in iot," *Evolving Systems*, vol. 12, pp. 19–36, 2021.

[17] S. Bai, J. Z. Kolter, and V. Koltun, "Convolutional sequence modeling revisited," 2018. [Online]. Available: https://openreview.net/forum?id=rk8wKk-R-

[18] D. Eswaran, C. Faloutsos, S. Guha, and N. Mishra, "Spotlight: Detecting anomalies in streaming graphs." New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3219819.3220040

[19] R. Paudel and W. Eberle, "Snapsketch: Network representation approach for anomaly detection in dynamic network," in *ACM MLG '20*, January 2020.