Scalable Simulation Algorithm to Generate Large-Scale Botnet Dataset using Role-Mining and Markov Chain

Tanveer Hossain Bhuiyan

tanveer.bhuiyan@utsa.edu

The University of Texas at San Antonio

Sarah Harun

Mississippi State University

Song Zhang

Mississippi State University

Hugh Medal

University of Tennessee at Knoxville

Research Article

Keywords: Botnet simulation, scalable algorithm, community structure, real-life dataset, parallel computing

Posted Date: August 16th, 2023

DOI: https://doi.org/10.21203/rs.3.rs-3246819/v1

Additional Declarations: No competing interests reported.

Scalable Simulation Algorithm to Generate Large-Scale Botnet Dataset using Role-Mining and Markov Chain

Tanveer Hossain Bhuiyan (Corresponding author)

Department of Mechanical Engineering, The University of Texas at San Antonio, TX, USA

Email: tanveer.bhuiyan@utsa.edu

Tel: (210) 458-5567

Sarah Harun

Department of Computer Science and Engineering, Mississippi State University, MS, USA.

Email: sarah.harun.06@gmail.com

Tel: (662) 471-2534

Song Zhang

Department of Computer Science and Engineering, Mississippi State University, MS, USA.

Email: sz49@msstate.edu

Tel: (662) 325-7510

Hugh Medal

Department of Industrial & Systems Engineering, University of Tennessee, TN, USA.

Email: hmedal@utk.edu

Tel: (865) 974-7647

Abstract

Detection of malicious networks (botnet) is becoming a major concern as they pose a serious threat to network security. But, botnet detection methods often perform very poorly in real-life datasets as the methods are not developed based on a real-life botnet dataset. A crucial reason for the detection methods not being developed based on a real-life dataset is the scarcity of large-scale, real-life botnet datasets. Due to security and privacy concerns, organizations do not publish their real-life botnet dataset. Realizing the need for a real-life large-scale botnet dataset, in this paper, we develop a simulation methodology to simulate a large-scale botnet dataset from a real-life botnet dataset. This simulation methodology is based on the Markov chain and role-mining approaches. Besides simulating the degree distribution, our simulation methodology also simulates triangles (community structures). We propose a novel scalable algorithm using parallel computing that generates large-scale botnet graphs from a small-size input dataset. To evaluate the performance of our simulation methodology, we compare our simulated graph with the original graph and with the graph simulated by the Preferential attachment (PA) algorithm based on the distributions of triangles, indegrees, and outdegrees. Results demonstrate that the distributions of the simulated graph generated by our methodology are very similar to the distributions of the original graph with minor real-life random variations. Results also demonstrate that our simulation algorithm substantially outperforms the PA algorithm in simulating the distributions of triangles and botnet subgraphs. To emphasize the accuracy of botnet simulation more, we provide a separate comparison between the botnet subgraphs of the simulated and the original graphs that demonstrates the similarity of our simulated botnet subgraphs with the original botnet subgraph. A comparison of our simulated scaled-up graph with the original graph demonstrates that our methodology preserves the triangle distribution and the botnet subgraphs of the original graph, whereas the PA algorithm fails to preserve the triangle distribution and the botnet subgraphs in the scaled-up graph.

Keywords: Botnet simulation, scalable algorithm, community structure, real-life dataset, parallel computing.

1 Introduction

The botnet dataset is a network dataset that contains information about the internet and intranet traffic with normal, malicious, and background activity. The main purpose of a botnet dataset is to provide an overall idea of the real effects of cyber attacks over a network so that researchers in network security and other relevant fields can study the dataset to find regular

patterns and abnormal behaviors, and/or test new methodologies. For this reason, the dataset must behave as realistically as possible concerning both normal and malicious traffic (Shiravi et al., 2012). Many existing graph theories, anomaly detection, classification, and clustering methods fail to correctly detect botnets from a real-life botnet dataset because of high-class imbalance, mixture and overlapping classes, noises, outliers, and real-life uncertainties involved in the dataset. Therefore, researchers need more realistic datasets to conduct experiments and evaluate their algorithms. But, one major obstacle to working with real-life network data is the lack of publicly available botnet datasets. Due to privacy and security concerns, organizations usually do not make their datasets publicly available. Though there are some publicly available datasets, they have several limitations. The main drawbacks of publicly available botnet datasets to date that limits their applicability in network security research are the following: (1) publicly available botnet datasets are created in an artificial environment, (2) a real dataset is modified to erase private information, (3) most of the datasets are not labeled, and (4) format of the dataset is quite complex and require documentation and extensive amount of time to preprocess. Therefore, there is a necessity for a simulation algorithm that can generate a real-life network dataset with botnet traffic, while preserving the privacy of the original network. Organizations can simulate a realistic dataset from their original dataset using the simulator and make the simulated dataset publicly available for research in the field of network security.

Besides the scarcity of publicly available botnet datasets, large-scale graph problems are also introducing new challenges to the research community in the last two decades (Zhang et al., 2018; Sarma et al., 2012; Leskovec et al., 2014; Klauck et al., 2015). Many existing botnet detection algorithms are becoming useless in terms of runtime or memory issues as they were not designed to handle large-scale graphs. In this regard, there is a necessity for a simulation algorithm that can generate a large-scale real-life network dataset with botnet traffic while preserving the privacy of the original network. This will enable the researchers in network security to develop new botnet detection algorithms for large-scale botnet datasets.

Therefore, we develop a simulation methodology based on the Markov chain and role-mining that can simulate a large-scale botnet dataset from a given (real-life) botnet dataset with similar degree distributions and community structures. We prioritize the accuracy of simulating botnet subgraph—a subgraph where only the bot (machines performing malicious activities connected via internet networks) nodes act as source nodes. The primary goals of this research are to (1) develop a simulation methodology capable of simulating large-scale botnet dataset accurately

while preserving the security and privacy of the original dataset; (2) develop a scalable algorithm to scale up the simulated graph; (3) evaluate the performance of the simulation algorithm by providing a comparison of the botnet subgraphs, overall graphs, and the scaled-up graphs; and (4) assess the performance of the simulation methodology compared to the existing simulation algorithms.

1.1 Related Literature

Graph simulation means generating a graph that possesses the features, characteristics, and patterns of a given graph. Despite the existence of several graph patterns in the literature, new graph patterns are still emerging. As numerous research fields are now using the graph as data and the size of graphs is increasing, more new graph patterns are becoming visible. Therefore, in this section, we discuss a few relevant graph properties first, following a review of literature on the existing graph simulators and botnet simulators. As we use the *role-mining* approach—a method closely related to clustering—in developing our simulation method, we discuss existing research on the role-mining method as well.

1.1.1 Graph Laws and Patterns

Despite some dissimilarities among graphs, some similar patterns among them are often observed. These patterns together characterize naturally occurring graphs. We discuss three common patterns often found in real-life Netflow graphs as follows:

Power-law: In statistics, a power law is a functional relationship between two quantities, where one quantity varies as an exponent of another. The degree distribution of a graph is a power law if the number of nodes N_d of degree d is given by $N_d \propto d^{-p}$, where $p > 0$ is called the power-law exponent (Newman, 2005). Graphs that possess power-law degree distributions are known as scale-free graphs. For example, the power-law pattern of indegree distribution of the scenario 5 of CTU-13 dataset (Garcia et al., 2014)—a publicly available real-life dataset—is shown in Figure 1. Skewed distributions, such as power laws have been found in several diversified phenomena including but not limited to the study of moon craters (Neukum and Ivanov, 1994), internet, web and online social networks (Faloutsos et al., 1999; Chakrabarti et al., 2004; Barabási and Albert, 1999; Newman et al., 2002), citation graphs (Redner, 1998), biological taxonomy (Willis and Yule, 1922), sales data (Kohli and Sah, 2003), and natural language processing (Siddharthan, 2002). Though it is an established and most commonly observed pattern in real-

life graphs, deviations from the power-law pattern are also found in several real-life phenomena (Pennock et al., 2002; Bi et al., 2001; Amaral et al., 2000).

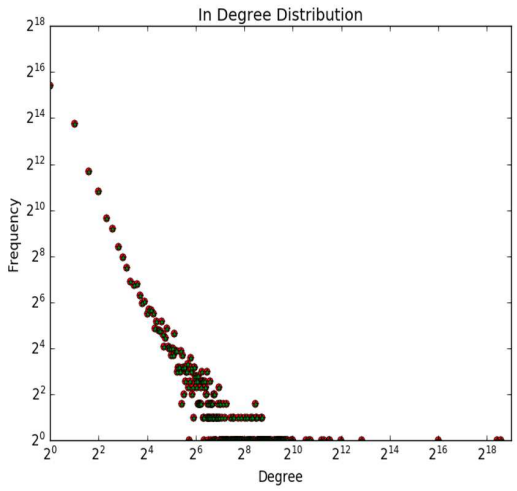


Figure 1: Power law in indegree distribution of CTU-13 scenario 5 (log-log scale).

Community Structure: A community is generally considered to be a set of nodes that are closer to each other than to nodes far from them. Figure 2 demonstrates three communities, where nodes having similar properties belong to the same community. This structure is usually found in many real-world graphs, especially on social networks. Flake et al. (2000) found communities of web pages on the World wide web (WWW). Communities based on race and age in a network of friendships in the United States are found by Moody (2001). Detection of communities is considered to be very important in a variety of research, mainly because each community represents an individual work unit of the overall system. For example, in metabolic networks, communities correspond to cycles or pathways; in protein interaction networks, communities correspond to proteins with similar functionality inside a biological cell; in citation networks, each community represents a research topic. Identification of these sub-structures within a network can provide insights into how networks function and topology affect each other (Girvan and Newman, 2002). Another important reason to consider communities is that they often have very different properties than the average properties of the networks. Thus, only concentrating on the average properties usually misses many important and interesting features of a network.

Triangle is the basic topological structure and a major metric that represents communities. The information about triangles is usually represented in terms of clustering coefficients which

are widely used in the literature (Chakrabarti and Faloutsos, 2006). Let G be a simple undirected graph with n vertices and m edges, where T and W denote the number of triangles, and wedges (a path of length 2), respectively. The global clustering coefficient of graph G is denoted as, $C = 3T/W$, which measures how often friends of friends are also friends in a social network.

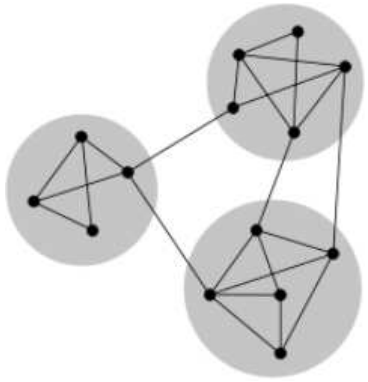


Figure 2: A small network displaying community structure with three groups of nodes.

Graph Diameter or Small-World Phenomena: The diameter of a network or graph is the maximum number of edges required to connect any pair of nodes in that graph. Most real-life graphs exhibit a relatively small diameter phenomenon, also known as the "small-world" phenomenon, or "six degrees of separation" (Travers and Milgram, 1967). The calculation of diameters is affected by nodes that are disconnected or outliers. Therefore, the effective diameter is considered to be a more reliable pattern. Effective diameter or eccentricity is the minimum number of hops in which some fraction (e.g., 80%) of all connected pairs of nodes can reach each other (Tauro et al., 2001). The effective diameter seems to be small for real-life large graphs, for example, 19 for the Web, 6 for the social networks, and around 12 for the router-level graph of the Internet (Leskovec et al., 2005b; Travers and Milgram, 1967; Faloutsos et al., 1999; Albert and Barabási, 2002). Besides the above-mentioned patterns, there exist several other patterns in the literature on graph analytics (Chakrabarti and Faloutsos, 2006).

1.1.2 Existing Graph Simulators

Graph simulation or generation is an interesting and challenging area for the research community in theoretical computer science, physics, and mathematics. There are several graph simulators in the existing literature.

Erdos–Renyi (ER) model (Erdős and Rényi, 1961; Erdos and Rényi, 1960) is considered to be the origin of graph simulation. It provides the first and simplest model for generating a random graph. This model can be defined as $G(n, p)$, where n and p are the numbers of nodes and the equal probability of edges to be present in the graph, respectively. But, as the ER model was not designed to simulate real-life graphs, it fails to simulate many desired properties of real-life graphs, such as power-law degree distribution, diameter, and eigenvalues.

Most of the recent works on graph simulation belong to the category known as *Preferential Attachment* (PA). Barabasi-Albert (BA) (Barabási and Albert, 1999) is the first of this category where the idea is to keep adding new nodes in the graph and prioritize the higher degree nodes while making a new connection. This simple idea leads to power-law tails in the degree distribution. However, the diameter in this model grows slowly with the number of nodes, which violates the "shrinking diameter" property mentioned in Section 1.1.1. Some alternate versions of BA are the following: modifications (Bu and Towsley, 2002), rewiring mechanism (Albert and Barabási, 2000), copying mechanism (Kleinberg et al., 1999), and "winners don't take all" model (Pennock et al., 2002). There are a few other generators (Kumar et al., 1999; Aiello et al., 2002, 2000) that simulate the degree distribution but fail to simulate other properties.

Another category of graph-generation method seeks to simulate the small diameter property of a real-life graph. For example, the "small-world" generator (Watts and Strogatz, 1998) and the Waxman generator (Waxman, 1988). BRITE (Medina et al., 2000) graph generator combines the idea of both the BA (Barabási and Albert, 1999) and the Waxman model (Waxman, 1988). The R-MAT (Chakrabarti et al., 2004) generator, belonging to this category, is one of the major breakthroughs in graph simulation literature. R-MAT can simulate power-law and exception of power-law (Pennock et al., 2002). R-MAT is scalable and works for the directed, undirected, bipartite, and weighted graph using the same approach. Notably, it has been selected as the generator for the Graph 500 Supercomputer Benchmark (Murphy et al., 2010). Though R-MAT has some desired properties (Leskovec et al., 2010), it is not successful in simulating community effect, especially the high clustering coefficients (Sala et al., 2010).

Kronecker graph generation (Leskovec et al., 2010; Leskovec and Faloutsos, 2007; Leskovec et al., 2005a) is based on the matrix operation, the *Kronecker product*. This generator can simulate heavy-tailed distributions for in-degree, out-degree, eigenvalues, and eigenvectors. The researchers also demonstrate how a Kronecker graph can match the behavior of several real-life networks, such as social networks, citations, the web, and the internet. But, as the Kronecker

algorithm is not designed to simulate topological structures, it usually fails to simulate the community structures, especially for sparse graphs.

Though the above-mentioned models may produce heavy-tailed degree distributions, the clustering coefficients of their simulated graph are often low (Sala et al., 2010). For this reason, none of these models explain community structure, which is one of the most striking features of interaction graphs.

There exist several other models that particularly aim to simulate the community structure or in other words, the clustering coefficient. Guo and Kraines (2009), Bansal et al. (2009), Newman's algorithm (Newman, 2009), and Gleeson's algorithms (Gleeson, 2009) belong to this category. Guo and Kraines (2009) produces a random graph by following power-law degree distributions and then keep rewiring the edges to produce triangles. They continue the process as long as it does not achieve the desired average clustering coefficients. This method fails to simulate local clustering coefficients and thus the topological structures of the graph.

Bansal et al. (2009) can be considered an updated version of Guo and Kraines (2009). They also rewire edges, but based on different criteria than Guo and Kraines (2009). The process of Bansal et al. (2009) continues until it reaches the desired global clustering coefficient. Newman's algorithm (Newman, 2009) seeks to simulate local triangular structures using the number of triangles and the number of single edges to which each node is involved as a parameter. But the problem of Newman's algorithm is that it over-uses the edges to produce the same number of triangles as in the real network. Gleeson's algorithm (Gleeson, 2009) generalizes Newman's model by starting from another higher-order motif – a k clique, which is a complete graph among k nodes, each of which is connected to every other node in the graph. This algorithm is expensive in terms of memory and CPU time.

One of the latest works on simulating clustering coefficient is BTER (Seshadhri et al., 2012). This algorithm hypothesizes that any graph with a heavy-tailed degree distribution and community structure contains a scale-free collection of dense Erdos–Renyi subgraphs. BTER used a combination of Erdos-Renyi and Chung-Lu model (Chung and Lu, 2002a,b). This approach mostly works with dense graphs and is not suitable for sparse graphs.

A few studies present simulation tools for botnet behavior [e.g., Kotenko et al. 2010; Vignau et al. 2021], but their focus is on the interaction between bots and defense strategies on a defender-attacker setting without considering the network topology. Kotenko et al. (2010) presents an agent-based simulation tool for the interaction of the botnet with the defense strate-

gies in an attacker-defender setting. Vignau et al. (2021) presents a simulation tool for predicting the interaction among different bots and how different botnet design strategies/features affect the spread of botnet in a dynamic competitive environment, where multiple bots compete for the same target. The authors ignore the network topology and the critical graph properties (e.g., community structure), considering only a set of potential targets for the bots. These studies do not simulate the botnet behavior to generate a large-scale, real-life, and class-imbalanced (i.e., the proportion of botnet traffic is very small compared to normal traffic) botnet dataset that contains both normal and botnet traffic.

1.1.3 Role-Mining

Role-mining or role-discovery is a new concept compared to community detection for network data. Recently, Rossi and Ahmed (2015) provides a descriptive and well-organized survey on all the existing role discovery techniques in the literature.

According to RolX (Henderson et al., 2012), there is a small difference between role-mining and role-discovery. Role-discovery is the assignment of different nodes in a graph to different roles, whereas role-mining concerns whether different nodes are allowed to access different roles in a controlled environment. But both techniques use similar algorithms for clustering the nodes into roles. Role-discovery, and role-mining are used interchangeably in the literature. In this research, we use the term role-mining though we do not consider access permission.

In RolX (Henderson et al., 2012), the authors extracted several features for each node and created a large 2D feature matrix. The non-negative matrix factorization (NMF) method was used to decompose the feature matrix into the following two matrices: node by role and role by feature. RolX showed that it is able to put similar nodes in the same role where each role has distinct characteristics. The authors used these roles for graph matching, sense-making, and transfer learning. RolX is linearly scalable with the number of edges.

Somaiya et al. (2010) used a Bayesian framework for role clustering and developed a Markov Chain Monte Carlo (MCMC) based algorithm that allows a single node to participate in multiple roles. This approach is not scalable for large graphs. Gilpin et al. (2013) studied the possibility of using external requirements for role discovery. The authors developed a framework, Guided Learning for Role Discovery (GLRD), that models the role detection problem as a constrained NMF problem. The authors improved the efficiency of this framework by using the least square formulation rather than optimizing the whole matrices. Ruan and Parthasarathy (2014) devel-

oped a novel algorithm, RC-Joint, to detect both community and roles from a network. Soft k-means were used on structural features to identify the roles. Rossi et al. (2013) developed a role detection method for a dynamic environment, where a series of network snapshots are available. This technique detects roles for each snapshot first and then calculates the transition of roles over snapshots. Next, they analyze the temporal pattern of role-changing for each node. This technique is applicable in anomaly detection and nodal behavior prediction.

1.2 Research Gap

Since around 2010, the scarcity of an effective botnet dataset has been reported in the literature (Shiravi et al., 2012; Rossow et al., 2012). Due to the lack of available and convenient real-life, large-scale botnet datasets, researchers in network security are struggling to evaluate the existing and new algorithms. Therefore, the need for a real-life, large-scale botnet dataset is becoming more evident. Moreover, varieties of botnets exist that demonstrate different botnet behavior. Successful detection of varieties of botnets requires studying different botnet datasets. Therefore, the need for different real-life, large-scale botnet datasets is crucial.

Though existing studies suggested several characteristics of an ideal botnet dataset, the process to capture such a dataset faces several trade-offs, such as ensuring data privacy while keeping transparency (Rossow et al., 2012; Shiravi et al., 2012). Because of security and privacy concerns, many organizations cannot publish original real-life graphs. Therefore, there is a clear need for a simulation algorithm that can simulate a large-scale NetFlow dataset containing botnet traffic similar to a real-life input dataset.

Garcia et al. (2014) tried to ensure all the characteristics mentioned by Rossow et al. (2012) and Shiravi et al. (2012) in their CTU-13 Netflow dataset, which is a real, labeled botnet dataset. But the CTU-13 dataset is smaller compared to today's large-scale botnet dataset. Because of its real-life characteristics, the CTU-13 dataset can be used as a seed graph for a graph simulator, but the simulated graph needs to be scaled up to create a large-scale dataset. This triggers the need for an efficient scalable graph simulation algorithm to generate real-life, large-scale graphs with botnet traffic regardless of a specific botnet behavior. Despite there exists a few botnet behavior simulation tools [e.g., Kotenko et al. 2010; Vignau et al. 2021], their focus is not to generate a large-scale, real-life botnet dataset containing both normal and botnet traffic while maintaining critical graph properties. Therefore, to the best of our knowledge, there does not exist a scalable graph simulation algorithm for generating real-life, large-scale graphs with

botnet traffic.

Additionally, despite the existence of numerous graph simulation algorithms, there are no existing universal graph simulators to date that can perform well in all types of graphs. There exists a variety of simulation algorithms depending on the graph size, sparsity, density, directed, undirected, weighted/unweighted, and single-edge/ multi-edge. Though most graph simulators can achieve very good accuracy in simulating degree distributions, all the existing graph simulation algorithms are facing challenges in simulating topological structures such as triangles. Triangle is one of the most useful metrics representing the topological structures of a graph and measures social unity. Several important characteristics of many real-life applications can be explained by triangles, such as homophily (e.g., people become friends with those similar to themselves) and transitivity (e.g., friends of friends become friends) in social networks, spam detection, and finding common topics on the WWW in graph mining (Becchetti et al., 2008; Eckmann and Moses, 2002).

But, from Section 1.1.2, we see that all the existing graph simulators are having issues in solving the problem of simulating local triangles. Therefore, triangle simulation, or in other words, simulation of clustering coefficient is still an open problem that requires more research to be conducted.

The above research gaps suggest that more research is needed in developing scalable graph simulation algorithms that can simulate real-life, large-scale botnet datasets containing both botnet and normal traffic along with the topological structures similar to a given botnet dataset.

1.3 Contributions

To fill the above-mentioned gaps in the literature, we propose a scalable graph simulation algorithm that can simulate a large-scale graph containing both botnet and normal traffic. The simulator provides priority to the accuracy of simulating the botnet behavior compared to the normal traffic. Our simulation algorithm can accurately simulate triangles and the associated degrees of each node, thereby preserving the community structure. Specifically, we have made the following contributions in this paper: (1) developed a new simulation algorithm based on Markov chain and role-mining that simulates a NetFlow graph with both normal and botnet traffic as well as accurately simulates triangle and degree distributions, (2) developed a new scaling-up algorithm, Enterprise connection algorithm, suitable for a distributed system, (3) provided a comparison of the simulated botnet subgraph from our algorithm with the original

botnet subgraph and with the simulated botnet subgraph generated by the PA algorithm, (4) provided a comparison of the complete simulated graph containing both normal and botnet traffic with the original graph and with the simulated graph generated by the PA algorithm, and (5) compared the scaled-up simulated graph with the original graph and with the scaled-up graph generated by the PA algorithm.

2 Problem Description and Hypothesis

In this research, we study the problem of simulating a real-life, large-scale NetFlow dataset with both normal and botnet traffic. Our goal is to simulate a large-scale botnet dataset similar to a given real-life botnet dataset that preserves the indegree, outdegree, and topological structures (triangles) of each node while hiding the identity of the nodes in the original graph. We emphasize more on accurately simulating the botnet behavior.

We use a Markov chain-based simulation to maintain correlation while connecting nodes rather than some degree-based approach as in the PA algorithm. When making a new connection in the PA algorithm, priority is given to the node that has the highest degree. In case of multiple nodes having the highest degree, the PA algorithm breaks the tie by randomly choosing one of those nodes. This simple degree-based rule of the PA algorithm helps to simulate the power-law pattern. In this research, we assume that the connection between two nodes should not necessarily be based on the highest degree. Rather, it is important to consider the clusters (roles) to which the communicating nodes belong to accurately simulate communities. Therefore, we incorporate the probability of connection between the nodes from different roles. Markov chain provides a way of modeling this behavior, where the transition probability matrix represents the probability of the next connection between two roles. After selecting the destination role using the transition probability matrix, we use either the highest degree or average role neighbor degree (ARND) (discussed in Section 3.3) to select the destination node.

We use role-mining to make the Markov chain process computationally feasible for large graphs. Without role-mining, if we want to compute the transition probabilities of each node in a network with a Markov chain, the computation of state transition probabilities becomes computationally infeasible, even for a small network. Therefore, the transition probability matrix mentioned above represents the probability of connection between roles. In our simulation methodology, there is a separate scale-up algorithm to produce a large-scale botnet dataset.

We implement our methodology on a real-life botnet dataset named CTU-13. This dataset is the only real-life, publicly available labeled botnet dataset containing varieties of bots. For details of the dataset, we refer readers to Garcia et al. (2014).

2.1 Hypothesis

Our simulation methodology is based on the concept of role-mining (Rossi and Ahmed, 2015), which is a clustering approach that defines meaningful roles (clusters) of the nodes in a network based on the graph properties of the nodes. Our simulation approach is based on the hypothesis that nodes in a network are divided into several roles such as server, router, personal machine, university machine, and so on, where each role usually communicates with some other roles at a certain frequency. Therefore, there is a possibility that there exist some communication patterns between the bot role and other roles. For example, some roles communicate more with bot roles compared to other roles in the network.

We provide a simple example for better clarification of the concept of role-mining. Consider a network where the nodes are grouped into the following four roles: A, B, C, and D. In Figure 3, we draw a small part of the network with 8 nodes and 4 roles. Here role A has 4 communications with role B, 2 communications with role C, and no communication with role D. Therefore when simulating the outgoing edges for nodes in role A, we can use the connectivity information with role B, C, and D of the original graph to simulate the destination nodes.

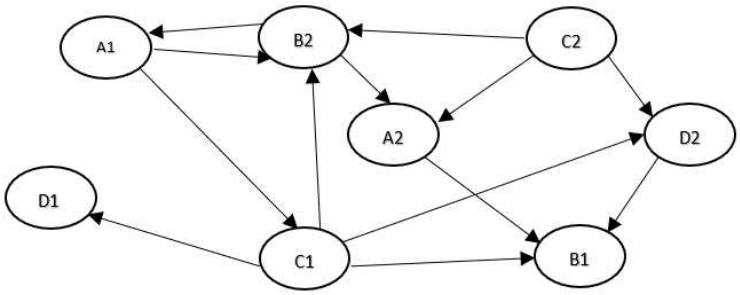


Figure 3: Communication among the roles in a small network.

3 Simulation Methodology

This section describes the steps of our proposed simulation methodology in detail. In the first step, we calculate some simple node features from a given graph. Next, we perform role-mining to group the nodes into roles. As our simulation procedure uses the role-mining (clustering) technique and the Markov chain to simulate the desired graph, we calculate several other properties related to roles and the Markov chain such as ARND and transition probability matrices. Then we simulate the triangles followed by the simulation of the remaining graph (remaining nodes and edges). The architecture of the complete simulation methodology is demonstrated in Figure 4.

We use the role-mining approach to enhance the computational speed and efficiency of the Markov chain in our simulation methodology. Dividing the nodes into roles speeds up the Markov chain-based simulation approach. In this research, role-mining can be considered an optional step as our simulation methodology needs to conduct the role-mining step only if the role information is not provided by the user. As the accuracy of role-mining is not the focus of this research, we use common graph properties and clustering algorithms to group the nodes into roles. If the role information is not given, we group the nodes using Gaussian mixture clustering algorithm based on their indegree, outdegree, and the number of triangles.

As one of our goals in this research is to generate a large-scale simulated graph similar to the original graph, we develop a scaling-up algorithm, the *Enterprise connection algorithm*, to simulate a large-scale graph in a distributed system utilizing parallel computing. The enterprise connection algorithm is independent of the simulation methodology and thus does not affect any of the steps of the simulation methodology. This scaling-up algorithm executes the steps of the simulation method in every processor and combines the results to produce a large-scale simulated graph. The enterprise connection algorithm is described in detail in Section 3.7.

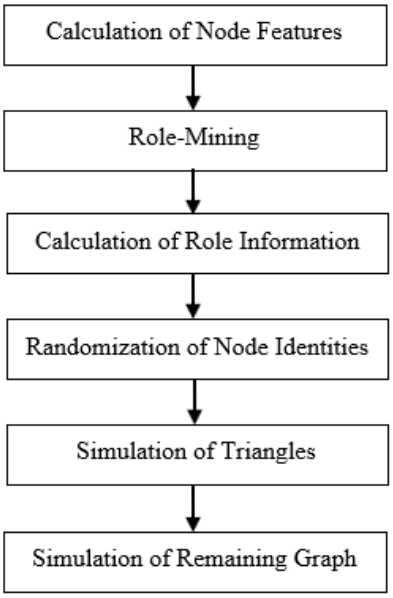


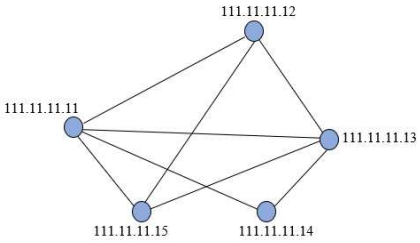
Figure 4: Architecture of the simulation methodology.

3.1 Calculation of Node Features

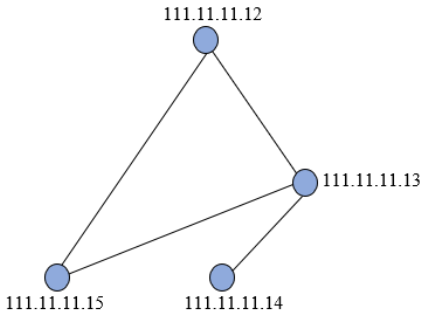
In this step, we calculate the four features—indegree, outdegree, number of triangles, and list of nodes involved in triangles—of each node from a given graph. These features (properties) of nodes are used in the role-mining and in the simulation process as parameters. When calculating indegree and outdegree, we consider the graph as directed and multi-edged (i.e., multiple edges between nodes). However, in triangle simulation, neighbors of a node are calculated considering the graph as undirected and single-edged. We make this simplifying assumption in triangle simulation as the definition of triangles in a multi-directed graph (i.e., directed graph with multiple edges between nodes) is quite complex. To our knowledge, there is no existing algorithm that can simulate triangles for a multi-directed graph.

The pseudo-code of calculating the node features is demonstrated in Algorithm 1. We discuss the procedure of calculating the number of triangles and list of neighbors involved in triangles using a small example graph demonstrated in Figure 5a with the corresponding properties of nodes provided in Table 1. There are five nodes in this example graph and each of them is involved in a different number of triangles. For each node that has at least one triangle associated with it, we list the neighbor nodes involved in those triangles and then remove the node as well as the edges connected to this node from the graph. This approach allows us to avoid

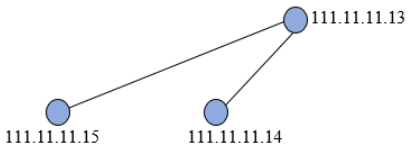
considering the same triangle three times for each of the associated nodes in counting neighbor nodes and simulating triangles. Also, this process gradually reduces the graph size that results in a gradual reduction of the computation time in calculating the list of neighbors involved in triangles. For example, after considering all the triangles for node 111.11.11.11 in Figure 5a, we remove the node 111.11.11.11 and the edges connected to this node from the graph. Therefore, the remaining graph becomes as shown in Figure 5b. Next, for node 111.11.11.12, we find out the list of neighbors involved in triangles from Figure 5b and remove the node 111.11.11.12 from the graph. Similarly, for node 111.11.11.13, the resulting graph is shown in Figure 5c.



(a) Sample small graph with triangles.



(b) Graph after removing node 111.11.11.11



(c) Graph after removing node 111.11.11.12

Figure 5: Calculation of list of neighbors involved in triangles.

Table 1: Properties of nodes in the graph shown in Figure 5a.

Row Id	**Node**	**Number of Triangles**	**List of Neighbours Participated in Triangles**
1	111.11.11.11	4	111.11.11.12, 111.11.11.13, 111.11.11.13, 111.11.11.14, 111.11.11.12, 111.11.11.15, 111.11.11.13, 111.11.11.15
2	111.11.11.12	3	111.11.11.13, 111.11.11.15
3	111.11.11.13	4	-
4	111.11.11.14	1	-
5	111.11.11.15	3	-

Algorithm 1 Calculate Node Properties.

```
1: function CALCULATE-PROPERTIES(G)
2:     properties ← Empty table
3:     undirG ← Undirected and single edge version of G
4:     tempG ← Copy of undirectedG
5:     rowId ← 0
6:     for each node n in G do
7:         neighbours ← FIND-NEIGHBOURS(tempG, n)
8:         Insert rowId,n.name,n.indegree(G),n.outdegree(G),n.triangles(undirG),neighbours
   in properties
9:         rowId ← increase by 1
10:        Remove n and its associated edges from tempG
11:    for each row r in properties do
12:        IDs ← Find rowIds in properties for each item in r.neighbours
13:        r.neighbours ← IDs
14:    return properties

15: function FIND-NEIGHBOURS(G, n)
16:    triangleNeighbours ← Empty list
17:    allNeighbours ← Neighbours of n in G
18:    for all combination of n1, n2 in allNeighbours do
19:        if n1, n2 has edge in G then
20:            Add n1 in triangleNeighbours
21:            Add n2 in triangleNeighbours
22:    return triangleNeighbours
```

3.2 Role-Mining

Our proposed simulation approach is based on the hypothesis that there are patterns in communications among the groups (roles) of nodes. As the main focus of this research is graph simulation using role information, we assume that the role information is provided by the user. Therefore, validation of role-mining is not the focus of this research. If there is any partial information available about the roles, we utilize it to define the roles as accurately as possible. Otherwise, we perform a simple role-mining algorithm to group the nodes into roles. For example, we use the CTU-13 Netflow dataset in our experimentation which does not have any role information. But, using previous bot detection studies on the CTU-13 dataset (e.g., Harun et al., 2017), we obtain the bot information that is used to group the bot nodes into a role. We group the remaining nodes into another 24 roles using Gaussian Mixture Algorithm. We choose this algorithm because of its simplicity, computational speed, and flexibility in dividing data into a desired number of roles. A Gaussian mixture model is a probabilistic model that assumes

all data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters. One can think of mixture models as generalizing k -means clustering to incorporate information about the covariance structure of the data as well as the centers of the latent Gaussians.

3.3 Calculation of Role Information

We calculate the transition probability matrices using the role information found in the previous step and the given graph (Netflow data). Transition probability matrices are part of the Markov chain. As it is computationally infeasible to calculate the transition probability matrix for each node, we compute the transition probability matrix (TPM) for connection between roles. We have one TPM for each role, where each row represents the current communication to another role. Each column represents the likelihood of next communication with another role. To provide better clarification, we explain this with a small example TPM shown in Table 2. Assume this TPM is for role A. The value of 0.8 in the second column of the first row means that if a node in role A is currently connected with another node in role A, there is 80% possibility that the next connection of a node in role A will be with a node in role B. In this way, we use the TPM to select the next destination role. Similarly, the value of 0.6 in the first column of the second row means that if a node in role A is currently connected with another node in role B, the likelihood that a node in role A will communicate with a node in role A is 60%.

Table 2: Transition probability matrix for role A.

	A	**B**	**C**
A	0.1	0.8	0.1
B	0.6	0.1	0.3
C	0.9	0.05	0.05

As mentioned earlier, we use an average role-neighbor degree (ARND) in simulating the connection. The ARND is computed when the bot role is acting as a source role. For example, when the bot nodes are acting as source nodes, assume that they communicate with 10 unique nodes of role A making 200 communications (edges), with 15 unique nodes of role B making 150 communications. Therefore, the ARND is [(A: 20), (B: 10)]. After choosing the destination role using the TPM, we use ARND if the chosen role is bot role; otherwise, we use the highest degree to choose the next destination node. We already stated earlier that we prioritize the simulation

of botnet behavior than the complete graph and this ARND helps us to achieve better results in simulating the botnet subgraph. For the remaining graph, we want to simulate the power-law pattern and therefore ARND is not useful for the rest of the graph. Botnet subgraphs, where only bot nodes act as source nodes, do not follow the usual power-law pattern. The pseudo-code to calculate these two properties are given in the Algorithm 2.

Algorithm 2 Calculate Role Information.

```
1: function CALCULATE-ROLE-INFORMATION(G, nRole, nodeRole)   ▷ nRole=Number of roles, nodeRole=Dictionary that contains role for each node
2:     roleTPM ← Dict()                ▷ Store the transition probability matrix for each role
3:     roleNodeDegree ← Dict()                                  ▷ Use to generate ARND
4:     for each role r in nRole do
5:         tpr ← array[nRole X nRole]
6:         rSource ← Use G and nodeRole to find netflows where r acted as source
7:         for prev ← 0, len(rSource) do
8:             curr ← prev + 1
9:             prevRole ← role of node at rSource[prev]
10:            currRole ← role of node at rSource[curr]
11:            tpr[prevRole][currRole] ← tpr[prevRole][currRole] + 1
12:            if prevRole is BotRole then
13:                roleNodeDegree[(prevRole, rSource[prevRole])] ← roleNodeDegree[(prevRole, rSource[prevRole])] + 1
14:        tpr ← divide each cell in a row by sum(row) in tpr
15:        roleTPM[r] ← tpr
16:    Use roleNodeDegree to calculate ARND
17:    return roleTPM, ARND
```

3.4 Randomization of Node Identities

In our simulation methodology, it is necessary to hide the identities of the original nodes in the simulated graph. There are two possible ways we can do that while preserving the degrees and community structures (triangles). One way is to rename the nodes; for example, if one node has an IP address of 148.23.161.11, we can rename it to a completely different IP address such as 111.12.76.198, or provide a random unique number such as 1891. This renaming can be done in any possible way as the user wants. Another more simple way is to shuffle the IP addresses of the nodes. If we are not scaling up the simulated graph, we choose to shuffle the node IP addresses because it keeps the node names more realistic. But, when scaling up the simulated graph to create a large-scale graph, we need to generate random names for each node in each processor instead of shuffling. Otherwise, the same IP address appears p times where p is the number of processor.

Whichever way we choose, this randomization step causes one problem in the property table (Table 1) calculated in Section 3.1. In Table 1, in the column *list of neighbours*, if we store the IP addresses of the nodes, then in this step after randomization, the list will be misrepresenting. To solve this problem, we provide an unique row id to each of the rows in the property table found after the calculation of node features step in Section 3.1. Instead of storing the IP addresses in the neighbor list, we store the unique row number of those neighbors. This approach solves the problem because simulation of triangles depends on the node properties—indegree, outdegree, and number of triangles—rather than the name of the nodes. An example of a property table after the calculation of node features step (i.e., before randomization) and after randomization of node identities step are given in Tables 3a and 3b, respectively.

Table 3: Randomize node identities.

(a) Before randomization.

Row ID	**Node**	**Number of Triangles**	**List of Neighbours Participated in Triangles**
1	111.11.11.11	4	2, 3, 3, 4, 2, 5, 3, 5
2	111.11.11.12	3	3, 5
3	111.11.11.13	4	-
4	111.11.11.14	1	-
5	111.11.11.15	3	-

(b) After randomization.

Row ID	**Node**	**Number of Triangles**	**List of Neighbours Participated in Triangles**
1	111.11.11.13	4	2, 3, 3, 4, 2, 5, 3, 5
2	111.11.11.15	3	3, 5
3	111.11.11.11	4	-
4	111.11.11.12	1	-
5	111.11.11.14	3	-

Therefore, after the randomization of node identities step, we have the following five parameters to pass to the next step of the simulation process: 1) randomized property table, 2) transition probability matrices, 3) ARND for bot role, 4) list of nodes belong to each role, and 5) number of netflows to simulate for each role.

3.5 Simulation of Triangles

As a first step toward simulating the Netflow graph with botnet traffic on it, we simulate the triangles. For the list of nodes that have a nonempty list of neighbors in the property table

(Table 3), we simulate triangles for each of them. For each node in that list, we select two consecutive neighbors from the list of neighbors and connect edges if no edges exist between them. Then we decrease the number of indegree, outdegree, and number of triangles for all three nodes accordingly. For each node, the process continues as long as there is neighbors in the list. The pseudo-code of this process is given in Algorithm 3. The output of this triangle simulation step is a small simulated graph that we pass to the next step. The next step adds more edges and nodes in the simulated graph.

Algorithm 3 Simulate Triangles.

```
1: function SIMULATE-TRIANGLES(properties)
2:     simuGraph ← Empty graph
3:     for each node x in properties that has triangle > 0 do
4:         for i ← 0, len(x.neighbours) − 1, 2 do
5:             y ← x.neighbours[i]
6:             z ← x.neighbours[i + 1]
7:             if x and y has no edge in simuGraph then
8:                 If x.outdegree > 0, add edge from x to y in simuGraph
9:                 If x.outdegree <= 0, add edge from y to x in simuGraph
10:            if x and z has no edge in simuGraph then
11:                If x.outdegree > 0, add edge from x to z in simuGraph
12:                If x.outdegree <= 0, add edge from z to x in simuGraph
13:            if y and z has no edge in simuGraph then
14:                If y.outdegree > 0, add edge from y to z in simuGraph
15:                If y.outdegree <= 0, add edge from z to y in simuGraph
16:            Decrease indegree, outdegree, triangles of x, y, z in properties accordingly
17:    return properties, simuGraph
```

3.6 Simulation of Remaining Graph

In this step, we simulate the remaining nodes and edges related to the botnet, normal, and background Netflows. The process starts by selecting each of the roles as source role, S. Then we start by selecting each source node A from role S. Denote the outdegree of A is A_d. As long as $A_d > 0$, we choose the destination role, D for the next communication using the transition probability matrix of that role. After selecting the destination role, D, we select the destination node B from that role using either the highest degree or ARND, depending on whether the source role is bot or not. If S is a bot role, we choose ARND in this step, otherwise we choose the highest degree to select B. When using ARND, we select a node that has a minimum degree mentioned in ARND. Next, $min(A_d, B_d)$ connections are made between the source node A and destination node B and the respective degrees are adjusted. Here, B_d is the indegree of B. This

process continues as long as there is netflows to simulate. The pseudo-code of this process is demonstrated in Algorithm 4.

Algorithm 4 Simulate Remaining Graph.

```
1: function SIMULATE-REMAINING-GRAPH(properties, nRole, roleTPM, simuGraph)
2:     for each role r in nRole do
3:         tpm ← roleTPM[r]
4:         srcNodes ← nodes from properties in role r and outdegree >= 1
5:         nNodes ← len(srcNodes)
6:         while nNodes > 0 do
7:             src ← random(srcNodes)
8:             prevRole ← generate a random role from nRole
9:             dst ← FIND-RANDOM-NODE-BY-ROLE(properties, prevRole, ARND)
10:            Add min(src.outdegree,dst.indegree) edge between src, dst in simuGraph
11:            Update indegree, outdegree of dst, src in properties
12:            while src.outdegree > 0 do
13:                randProb ← generate random probability
14:                sumProb ← 0
15:                for currRole ← 0, nRole do
16:                    sumProb ← sumProb + roleTPM[prevRole][currRole]
17:                    if sumProb > randProb then
18:                        prevRole ← currRole
19:                        dst ← FIND-RANDOM-NODE-BY-ROLE(properties, prevRole, ARND)
20:                        Add min(src.outdegree,dst.indegree) edge between src, dst in simuGraph
21:                        Update indegree, outdegree of dst, src in properties
22:                        break
23:            Remove src from srcNodes
24:            nNodes ← nNodes − 1
25:    return simuGraph

26: function FIND-RANDOM-NODE-BY-ROLE(properties, r, ARND)
27:    nodes ← Empty list
28:    if role is BotRole then
29:        nodes ← nodes from properties in r role and has indegree >= ARND[role]
30:    else
31:        nodes ← highest degree nodes from properties within role r
32:    if nodes is Empty then
33:        nodes ← nodes from properties which has indegree >= 1
34:    return random(nodes)
```

3.7 Enterprise Connection Algorithm (Scaling-up)

In this algorithm, we assume that our simulated large graph consists of many enterprises (subgraphs) with varying sizes, characteristics, and topology linked together to form a large network. To establish the connection between enterprises, we choose a few nodes with high degrees in each enterprise (indegree or outdegree) as they are likely to be servers. A small example of the enter-

prise connection algorithm is demonstrated in Figure 6. We see that there are five enterprises in Figure 6. A node with high degree is selected from each enterprise to connect the enterprises with each other. The red edges in Figure 6 represent the inter-enterprise connection.

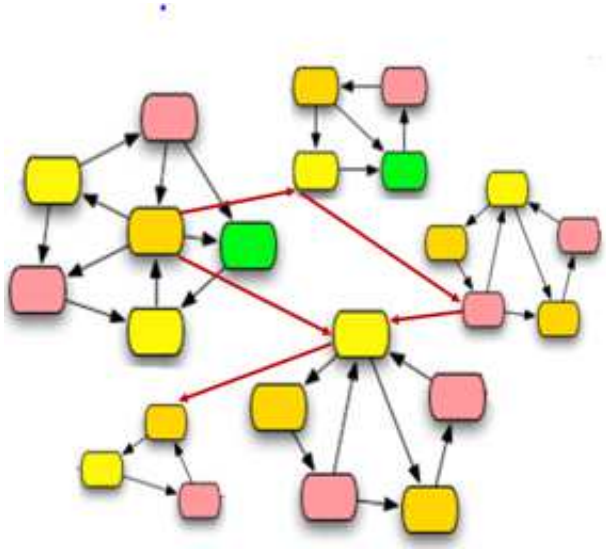


Figure 6: Connection between different enterprises.

The steps of our enterprise connection algorithm are presented in Algorithm 5. If our goal is to generate a simulated graph that is P times larger than the original graph, then this algorithm will execute right after the simulation of the remaining graph step (as described in Section 3.6) of our simulation procedure. In a distributed system, the enterprise connection algorithm requires P processors, where each processor executes the procedures described in Sections 3.2, 3.3, 3.4, 3.5, and 3.6. As the results from the role mining algorithm are different for each of the P processors, each of the simulated graphs generated by each processor will have random variations from each other. After generating simulated graphs in each of the processors, we have P different disconnected subgraphs (enterprises). Next, we choose N nodes with the highest indegree and N nodes with the highest outdegree from each of the subgraphs. We calculate the desired number of connections, E, among enterprises. We randomly choose a source node from the set of nodes with the highest outdegree and a destination node from the set of nodes with the highest indegree and add an edge between the selected nodes. This process continues until the desired number of connections (E) is reached. We can control E using another parameter called fraction, F, that ranges between 0 and 1.

Algorithm 5 Enterprise connection algorithm.

```
1: function ENTERPRISE-CONNECTION-ALGORITHM(F, N, P)
2:     Calculate node features
3:     Simulate individual enterprise using role-mining and the remaining steps in each processor
4:     N1 ← Select top N nodes with highest indegree from each enterprise
5:     N2 ← Select top N nodes with highest outdegree from each enterprise
6:     E ←F∗N∗P∗2
7:     for i ← 0, E do
8:         Randomly choose a source node from N2
9:         Randomly choose a destination node from N1
10:        Connect these two nodes
```

4 Results and Discussion

In this section, we evaluate the quality of our simulated graph and compare the similarity of the simulated graph to the original graph. We also compare the performance of our simulation methodology with the PA algorithm mentioned in Section 1.1.2. We evaluated our simulation methodology using the CTU-13 dataset. The CTU-13 Netflow dataset has 13 scenarios among which scenarios 5, 7, 11, and 12 are used in our numerical experiments to validate our proposed simulation methodology. Table 4 provides a summary of the number of nodes, edges, and bot nodes in each of these scenarios. We implement our methodology using Python 2.7 along with the following packages: Scikit-learn, OpenCV, Networkx, Numpy, and Pandas.

We designed the Enterprise connection algorithm in such a way that it is suitable for a parallel and distributed system. This algorithm is capable of generating a graph of size m where m is linearly scalable with the number of processors. Due to hardware limitations, we ran experiments on a personal computer with Intel Core i5 @2.2 gigahertz with 8 gigabyte of installed RAM. We used three processors to test the scalability of the Enterprise connection algorithm. However, with the availability of a larger number of processors, the Enterprise connection algorithm can generate much larger graphs.

The experimental results are organized in three major categories: (1) comparison of the simulated botnet subgraphs from our proposed algorithm and from the PA algorithm with the original botnet subgraph; (2) comparison of the overall simulated graphs from our proposed algorithm and from the PA algorithm with the overall original graph; and (3) comparison of the scaled-up graphs from our simulation algorithm and the PA algorithm with the original graph. It is to be noted that all simulated graphs—generated using our algorithm and the PA algorithm—are scaled up using the Enterprise connection algorithm discussed earlier in

Section 3.7. We demonstrate the comparison among botnet subgraphs by (i) comparing the graph statistics and (ii) visualizing the graphs. To compare the overall graphs, we compare the following results: (i) graph statistics, and (ii) histograms of the indegree, outdegree, and triangles. As the size of the overall graphs are much larger, comparison using a visualization approach is not very helpful for overall graphs. The graph statistics used in the comparison of the results are described in Section 4.1.

Table 4: Experimental dataset.

Scenario	**Nodes**	**Edges**	**Bots**
Sc5	41658	129832	1
Sc7	38204	114077	1
Sc11	41931	107251	3
Sc12	94434	325471	3

4.1 Graph Statistics Used in Comparison

Following are the graph statistics we use to compare the botnets and the overall graphs in the next sub-sections.

Average Degree:

Let $d(v_i)$ be the degree of vertexv_i and n be the number of vertices in total. The average degree of G is defined as:

$$d(G) = \frac{\sum_{1 \leq i \leq n} d(v_i)}{n}$$

Graph Diameter:

The diameter of a graph is the length $max_{u,v}d(u,v)$ of the "*longest shortest path*" between any two vertices (u,v) of the graph, where $d(u,v)$ is the shortest graph distance. In other words, graph diameter is the shortest distance between the two most distant nodes in the network.

Modularity:

Modularity is a measure of the structure of networks or graphs. It is designed to measure the strength of division of a network into modules (also called groups, clusters or communities). Networks with high modularity have dense connections between the nodes within modules but sparse connections between nodes in different modules. Modularity is often used in optimization

methods for detecting community structure in networks.

4.2 Comparison of Botnet Subgraph

In this section, we compare the botnet subgraphs simulated using our approach and the PA algorithm with the original botnet subgraph of CTU-13 Netflow dataset. Here, the term, *botnet subgraph,* means the subgraph where the bot nodes act as a source node. The comparison is conducted based on the graph statistics mentioned earlier and the visualization approach. The labels "original", "simulated", and "preferential" in figures in the following subsections denote the original botnet subgraph, simulated botnet subgraph using our proposed algorithm, and the simulated botnet subgraph using the PA algorithm, respectively.

4.2.1 Comparison of Graph Statistics

We use Gephi to calculate the number of nodes, average degree, and modularity of the botnet subgraphs. Graph diameter is used in the next section while comparing the overall graphs.

In Figure 7, we compare the number of nodes in the original botnet subgraph and in the simulated botnet subgraphs using our algorithm and the PA algorithm in all four scenarios of the CTU-13 dataset. For botnet subgraphs, the number of destination nodes is much higher than the number of source nodes. While simulating the botnet subgraph, the number of source nodes (bot nodes) is known but the number of destination nodes is unknown. We calculate the destination roles using the Markov chain and then find the destination nodes from that role using ARND. Therefore, the number of destination nodes in the simulated botnet graph may vary from the original botnet graph. We see from Figure 7 that the simulated botnet subgraphs using our algorithm have a similar number of nodes to the original botnet subgraphs in all four scenarios. On the other hand, only one (scenario 11) of the four botnet subgraphs simulated by the PA algorithm has a similar number of nodes to the original botnet subgraph. Though scenario 12 has more nodes compared to other scenarios, our simulation approach performed well in this scenario as in other scenarios. As the PA algorithm always prefers a node with the highest degree as the destination node, simulated graph generated by this algorithm has less variation in the number of destination nodes.

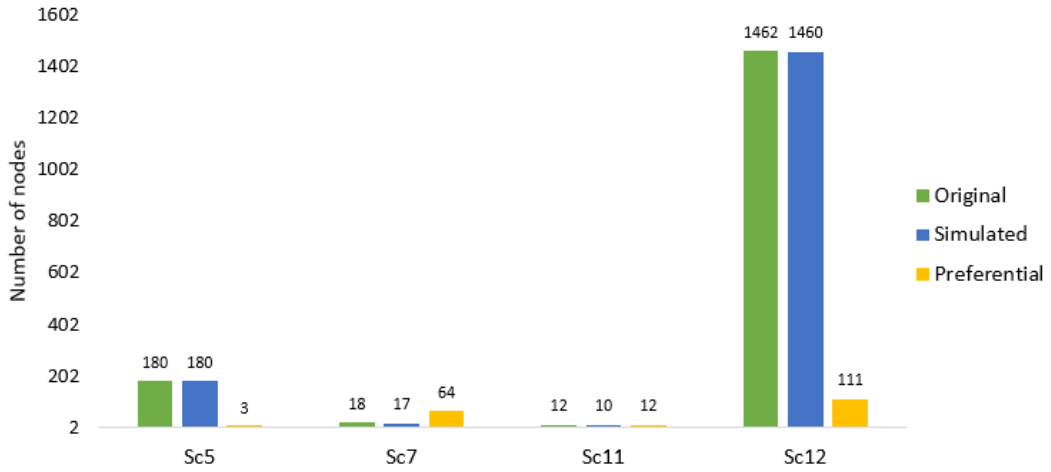


Figure 7: Comparison of the number of nodes in botnet subgraphs.

Figure 8 shows the average degree for the original botnet subgraph as well as the simulated botnet subgraphs using our algorithm and the PA algorithm in all four scenarios. For all scenarios of the CTU-13 dataset, the simulated botnet subgraphs using our algorithm have almost the same average degree as the original botnet subgraph, except for scenario 11. The botnet of scenario 11 has a comparatively higher number of edges compared to the number of nodes (the number of nodes and edges are 12, and 8164, respectively). Our simulated botnet has 10 nodes instead of 12 and that caused such differences in the average degree of the nodes in scenario 11. The PA algorithm achieved a similar average degree only in scenario 11 (dense graph) and failed to produce similar average degrees for scenarios 5, 7, and 12 (sparse graphs). It seems the PA algorithm performed well in simulating the botnet only for dense graphs.

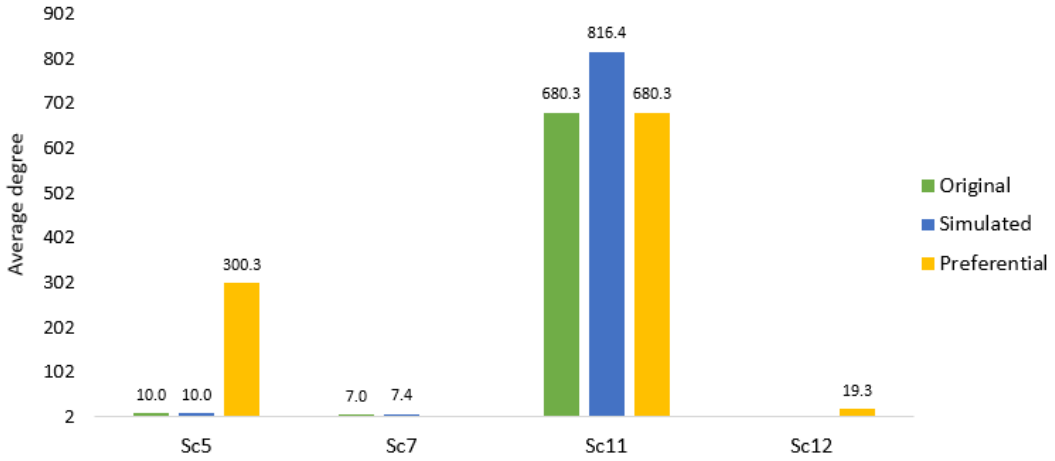


Figure 8: Comparison of the average degree of botnet subgraphs.

Modularity is the graph measure that is related to the community structures (clusters) in the graph. In Figure 9, we see that scenarios 5, 7, and 11 have modularity of approximately 0 in the original botnet and so as in the simulated botnet using our algorithm. Original botnet subgraph in scenario 12 has a high modularity value of 0.46 and the simulated botnet subgraph using our algorithm has a value of 0.47. Therefore, our simulation methodology performed well while simulating the clusters in the botnet. The yellow bars in Figure 9 demonstrate the modularity values found in the botnet subgraph generated by the PA algorithm. We see that the PA algorithm successfully simulates the modularity in scenarios 5 and 7, but failed to do so in scenario 12, even though three of these graphs are sparse graphs. The PA algorithm fails for the dense graph in scenario 11 as well. While connecting the nodes and edges, the PA algorithm does not consider anything related to communities or clusters. Therefore the communities (and modularity) generated by the PA algorithm might be random.

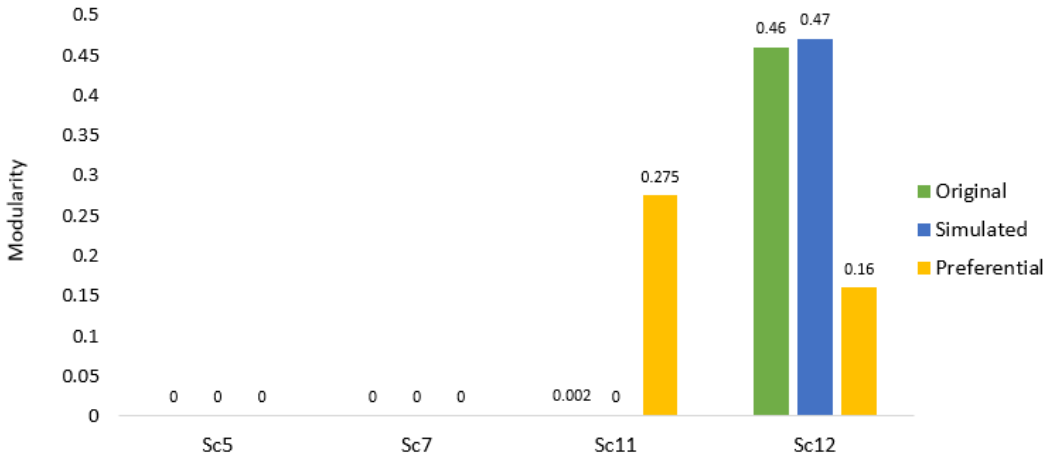


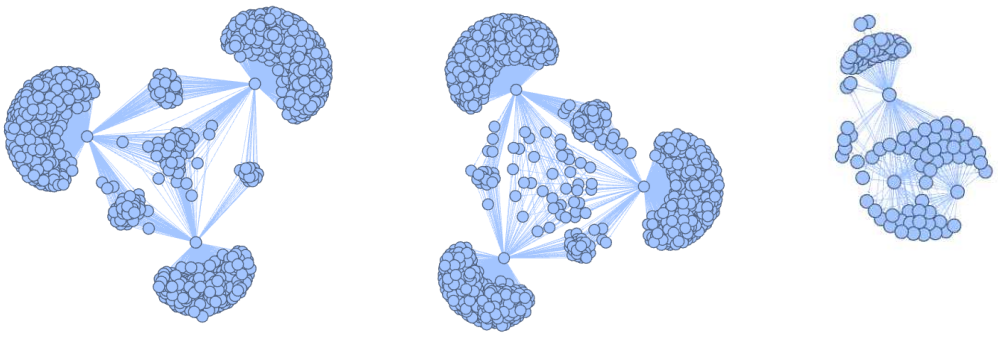
Figure 9: Comparison of the modularity of botnet subgraphs.

4.2.2 Comparison using Visualization

In this research, in addition to simulating communities and other properties for the overall graphs, our simulation methodology emphasizes especially simulating the botnet subgraph. The botnets of scenarios 5 and 7 are much smaller compared to the botnets of scenarios 11 and 12. There is only one bot in scenarios 5 and 7 and three bots in scenarios 11 and 12. In this section, we visualize and compare two botnets from scenarios 11 and 12 to visualize how similar the

simulated botnets from our algorithm and the PA algorithm are to the original botnets. We have drawn these graphs using the *ForceAtlas2* layout of Gephi.

From Figures 10a and 10b, we see that the main pattern of the original botnet and the simulated botnet from our algorithm are the same. The three big clusters on three ends are visible in both the original and the simulated graphs. But, the small clusters in the center area of the original graph seem a little bit different than the graph simulated by our algorithm. In the original graph, there are a few clusters in center area and nodes within each cluster are strongly connected. But in the simulated botnet, the nodes in the center area are scattered individually, or into several small clusters. These few small differences can be considered as normal randomness of a real-life dataset. Comparing the botnet subgraph generated by the PA algorithm with the original botnet subgraph as in Figures 10c and 10a, we see that the simulated graph using PA is very different than the original graph. As we construct three of these botnet subgraphs (original, simulated graphs using our algorithm and the PA algorithm) using the same graph layout algorithm (ForceAtlas2), the similarities and differences are quite visible at a quick glance.

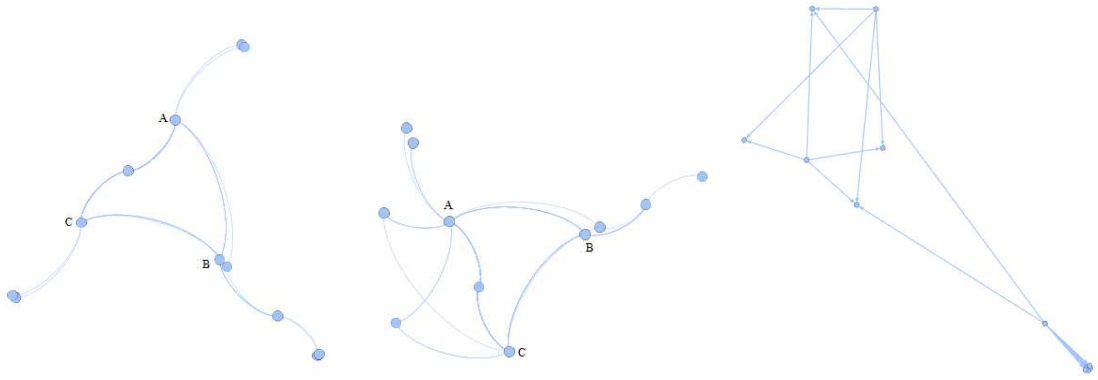


(a) Original botnet subgraph. (b) Simulated botnet by our algorithm. (c) Simulated botnet by the PA algorithm.

Figure 10: Visualization of scenario 12 botnet subgraphs.

Figure 11 demonstrates botnet subgraphs (original, simulated graphs using our algorithm and the PA algorithm) for scenario 11. If we compare our simulated botnet subgraph with the original botnet subgraph in Figures 11b and 11a, at first glance, the two graphs may look different, but basically they are very similar. They look different because of the way Gephi visualizes the graph. The same graph can be visualized in many different ways by changing the

node positions only. There is a major triangular shape in both the original and the simulated graphs defined by nodes A, B and C. In both graphs, A is connected with two other nodes besides the triangular connections. The same is true for nodes in the position of B. The only difference is the two nodes that are connected with C; in the original graph they are not connected with any of A and B, but in the simulated graph, they are connected with A. Now, comparing the botnet subgraphs generated by the PA algorithm with the original botnet subgraph in Figures 11c and 11a, we see that there is no similarity in the layout or community structures.



(a) Original botnet subgraph. (b) Simulated botnet by our algorithm. (c) Simulated botnet by the PA algorithm.

Figure 11: Visualization of scenario 11 botnet subgraphs

4.3 Comparison of the Overall Graph

In this section, we compare the overall simulated graphs using our simulation methodology and the PA algorithm with the original overall graph. Here, the term *overall graph* means the graph with botnet, normal, and background activities. We compare the simulated and the original graphs based on graph statistics and histograms of indegree, outdegree, and triangles. As the overall graph contains more than 100,000 nodes and 300,000 edges, a comparison using a visualization approach is not an effective way for the overall graphs. The labels "original", "simulated", and "preferential" in figures in the following subsections denote the original botnet subgraph, simulated botnet subgraph using our proposed algorithm, and the simulated botnet subgraph using the PA algorithm, respectively.

4.3.1 Comparison of Graph Statistics

We compute the graph statistics—average degree, modularity, and network diameter— for the original and simulated graphs to validate our proposed graph simulation methodology. As we already mentioned before, Gephi is used to compute all the graph statistics.

Comparison of the average degree among the original graph, simulated graph using our algorithm, and simulated graph using the PA algorithm are demonstrated in Figure 12. We see that the simulated graphs generated using our simulation approach have exactly the same average degree as the original graph in all four scenarios of the CTU-13 dataset. But, the graphs generated by the PA algorithm have similar average degrees for scenarios 5, 7, and 11 which are small graphs. The PA algorithm fails to produce a similar average degree for scenario 12, which is a comparatively larger graph in size.

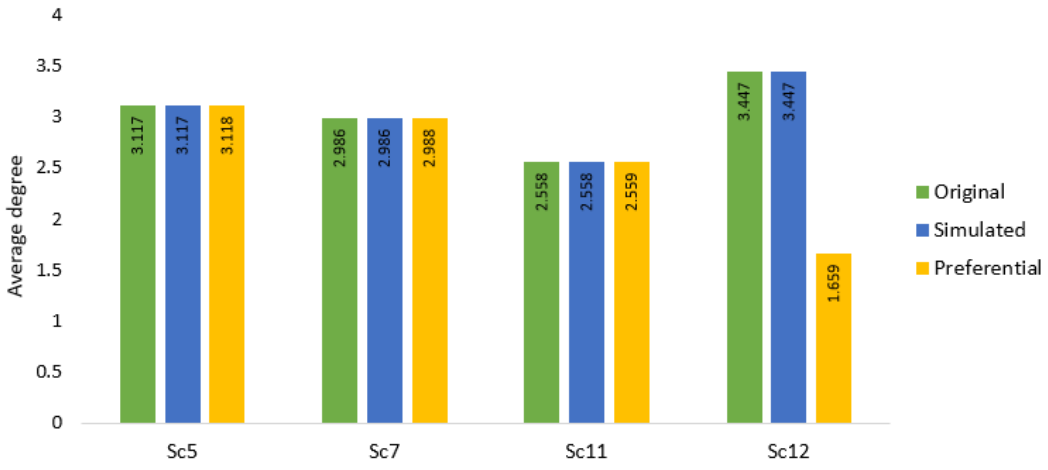


Figure 12: Comparison of average degrees of overall graphs.

As we mentioned before, modularity is related to community structures and thus it is related to triangles as well. Simulation of triangles or community structures is still an open problem that has not been solved yet. It is a hard and challenging problem and numerous research are ongoing to achieve better accuracy in simulating triangles. Our simulation technique also has one limitation while simulating triangles as mentioned earlier in Section 3.1 and 3.5. Despite that limitation, the modularity of our simulated graphs is very close to the values of the original graphs. The modularity of our simulated graph is almost the same as the original graphs for scenarios 5, and 11 and slightly higher for scenarios 7 and 12. The graphs generated by the PA

algorithm also performed well and achieved similar modularity values for scenarios 7 and 12 and slightly different for scenarios 5 and 11.

A point to note here is that CTU-13 graphs are quite sparse graphs and the number of triangles (communities) involved in each scenario (e.g. 5, 7, 11, and 12) is quite low compared to the number of nodes and edges involved in that scenario. Thus comparison of the overall graphs' modularity may not reflect properly how well each of these simulation algorithms performed in simulating each of the small communities. That's why we also present an in-depth comparison of the graph properties among original and simulated graphs using histograms in Section 4.3.2.

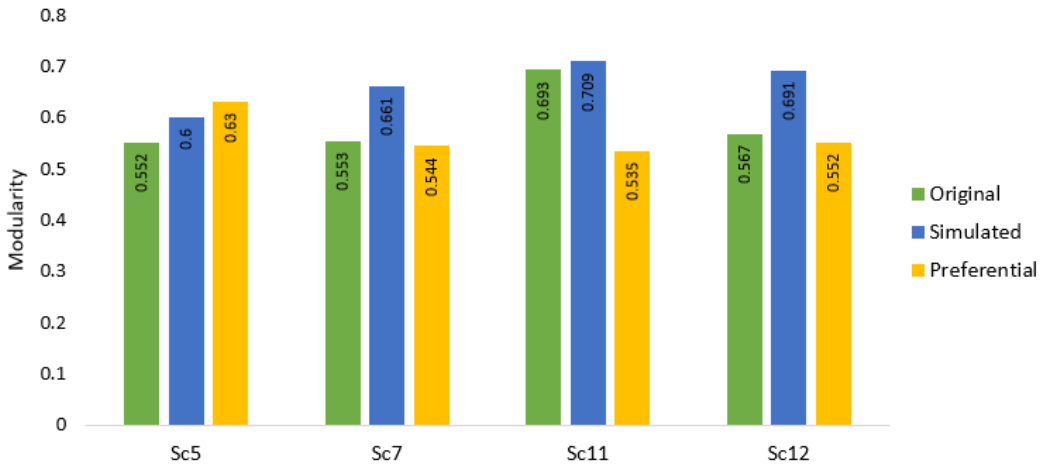


Figure 13: Comparison of modularity of overall graphs.

We see from Figure 14 that the diameter of our simulated graphs is similar to the original graph for scenarios 5, 7, and 11 and larger for scenario 12 only. The PA algorithm achieved similar values for scenarios 7, 11, and 12 but failed for scenario 5.

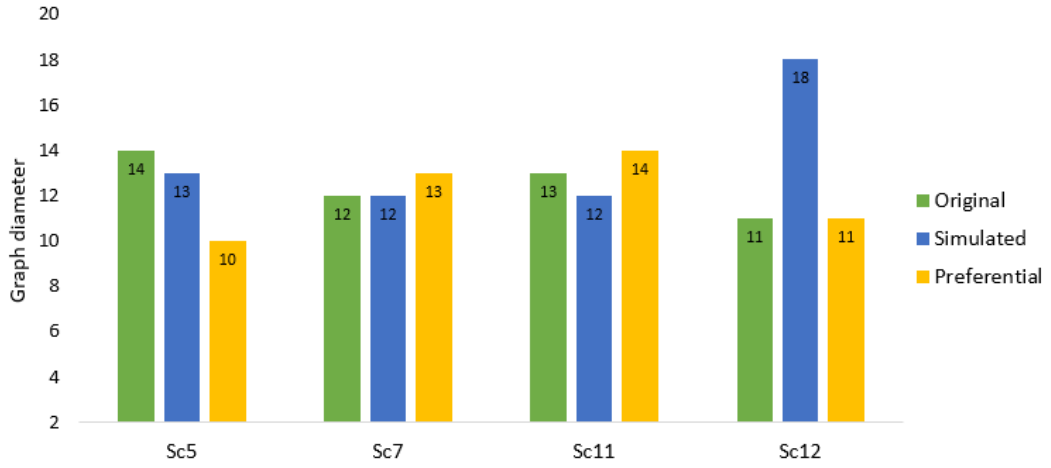
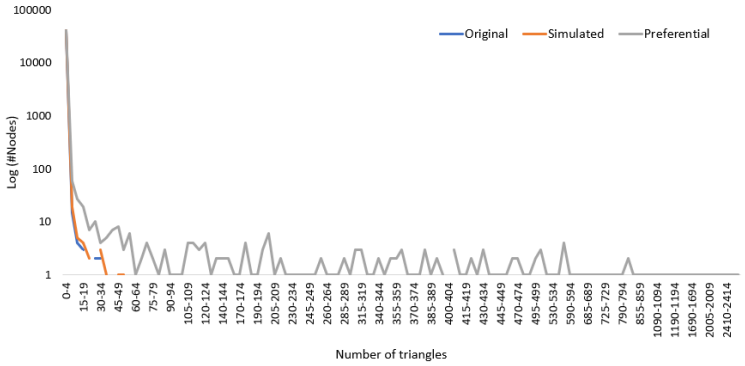


Figure 14: Comparison of network diameter of overall graphs.

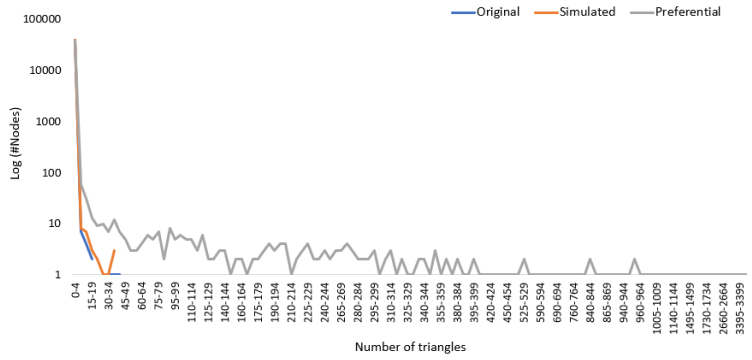
4.3.2 Comparison of Histograms

As a part of our validation process, we compare histograms of the overall graphs simulated by our simulation technique and the PA algorithm with the original graph of scenarios 5, 7, 11, and 12. We generate histograms for the indegrees, outdegrees, and triangle distributions of the original and the simulated graphs. The number of bins is different in the indegree, outdegree, and triangle histograms as it depends on the range of values.

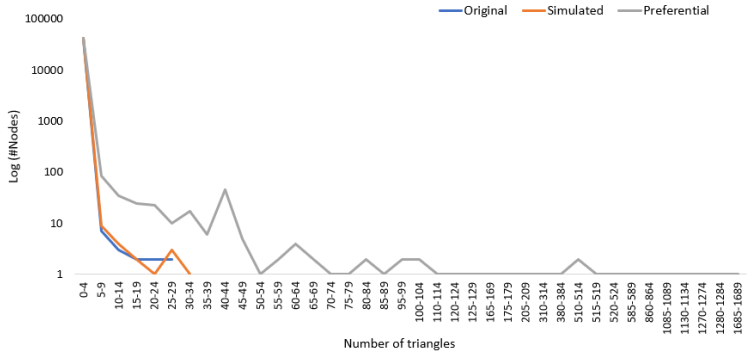
The triangle distributions are generated assuming the graph is undirected and single-edged. To compare the triangle distributions of the simulated graphs generated by our simulation methodology and the PA algorithm with the original graph, we plot a histogram in Figure 15 for scenarios 5, 7, 11, and 12. In Figure 15, bin width for the histogram is 5 and the number of nodes (y-axis values) is presented in a logarithmic scale to fit within the figure. We see from Figure 15 that the PA algorithm simulates lots of nodes with a large number of triangles that are not present in the original graph. Compared to the PA algorithm, our simulation algorithm simulates a much similar triangle distribution to the original graph. But, from this histogram, it is hard to understand how each of these simulation algorithms (Our algorithm and the PA) performed while simulating the nodes that are connected in triangles in the original graph. To understand that, we need to emphasize more the bins that have nodes in the original graph.



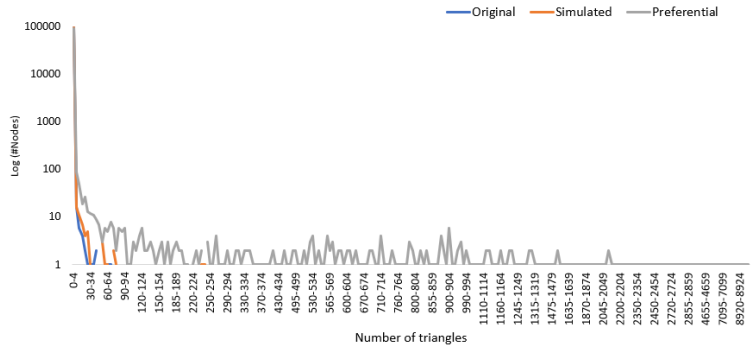
(a) Scenario 5.



(b) Scenario 7.



(c) Scenario 11.



(d) Scenario 12.

Figure 15: Comparison of full triangle histograms of overall graphs (original vs simulated vs preferential).

But the number of nodes involved with 0 (triangle-free) and one triangle is much higher (more than 99%) compared to other nodes. Therefore, it is not possible to visualize the values for all the bins by plotting a complete histogram in one figure. Though it is possible to demonstrate the values for all bins in a single figure using a logarithmic scale (similar to Figure 15), small differences between the simulated and original values are lost. In this research, it is important to account for the small differences in the number of triangles between the simulated and original graphs. Additionally, both the PA algorithm and our simulation methodology produce some nodes with a large number of triangles. As a result, histograms for the simulated graphs contain lots of extra bins compared to the original graph.

As the bins for 0 and 1 triangle dominate the overall histogram and make it difficult to compare the performance of the simulation algorithms on other bins, we present Figure 16 to compare the number of nodes that are not associated with any triangles and Figure 17 to compare the number of nodes that are associated with a single triangle. While simulating the nodes without any triangles in Figure 16, the performance of our simulation algorithm and the PA algorithm is almost similar. But, when comparing the performance while simulating the nodes with a single triangle in Figure 17, we see that the PA algorithm performed better than ours in scenarios 5, 7, and 12.

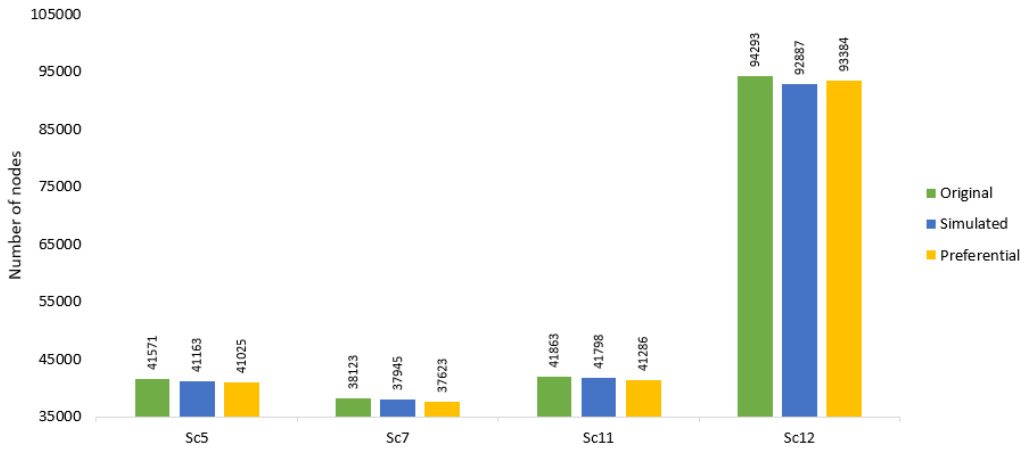


Figure 16: Comparison of the number of nodes without a triangle.

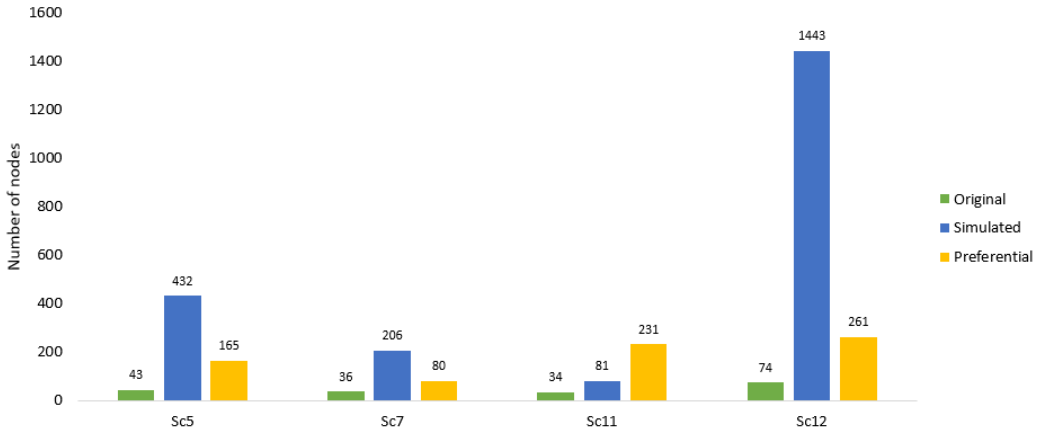
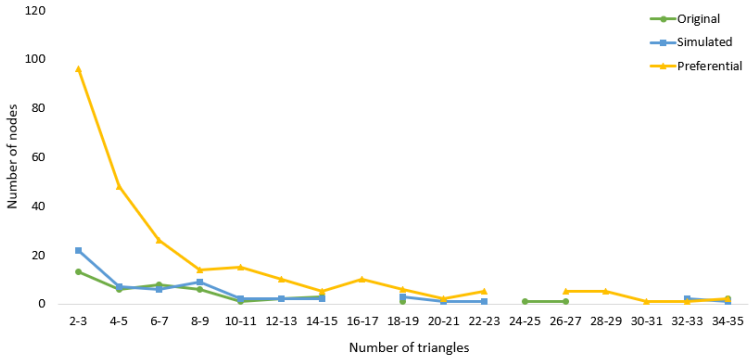
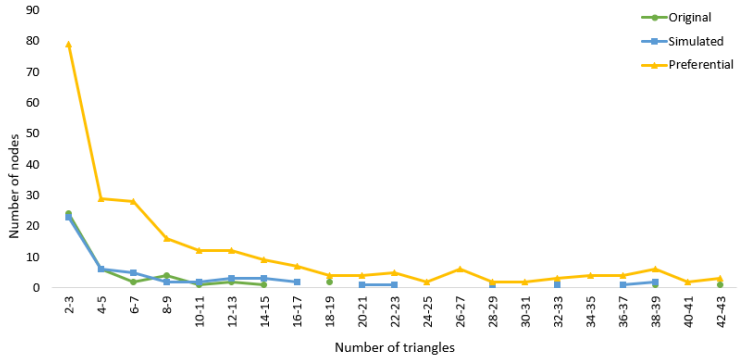


Figure 17: Comparison of the number of nodes with a single triangle.

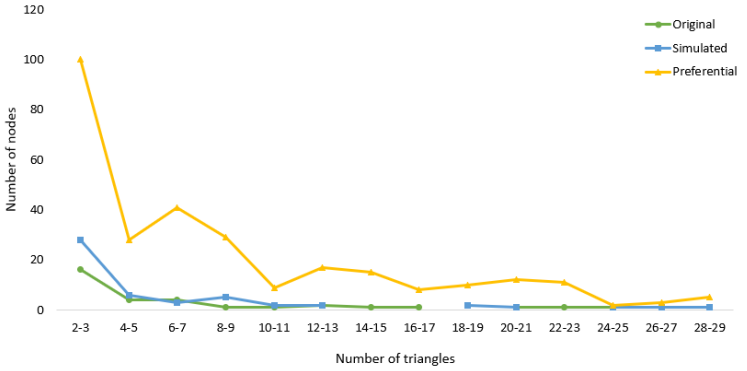
Now, in Figure 18, we only show the range of bins that are present in the original graphs to compare how the simulation algorithms performed in simulating nodes that are involved in larger communities in the original graphs. We removed the 0-1 bin as we already compared them. Figure 18 contains the histogram for scenarios 5, 7, 11, and 12. In all scenarios, our simulation methodology performed much better than the PA algorithm. One of our goals in this research is to develop a simulation methodology that can generate similar communities but not exactly the same communities. We expect to have random variations which makes it more realistic. We see from Figure 18 that our simulation algorithm successfully achieved similar distributions with some randomness as well.



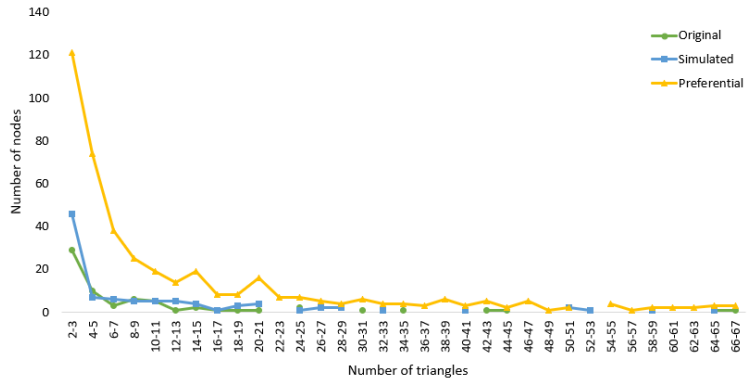
(a) Scenario 5.



(b) Scenario 7.



(c) Scenario 11.



(d) Scenario 12.

Figure 18: Comparison of triangle histogram for selective bins present in the original graph.

To evaluate the indegrees and outdegrees of the simulated graph, we generate histograms for indegrees and outdegrees for both the original and simulated graphs. Unlike the histograms for triangles, we visualize the values for all bins of the histograms of indegrees and outdegrees in a single figure. Figures 19 and 20 demonstrate the histograms for indegrees and outdegrees of the nodes in the simulated and original graphs for scenario 12. Similar results are found for scenarios 5, 7, and 11. As indegrees and outdegrees can vary within a large range of values, we use a logarithm scale while plotting the number of nodes. Bin width for these histograms is 30. From Figures 19 and 20, it is evident that our simulated graphs achieve almost similar degree distributions to the original graphs (for both indegree and outdegree). As the PA algorithm is designed to simulate degree distributions correctly, it achieved similar distribution successfully as well. It is to be noted that for some bins, there are no visible values. This is because the values corresponding to those bins are close to 1, which become 0 due to the application of logarithm (Figures 19 and 20).

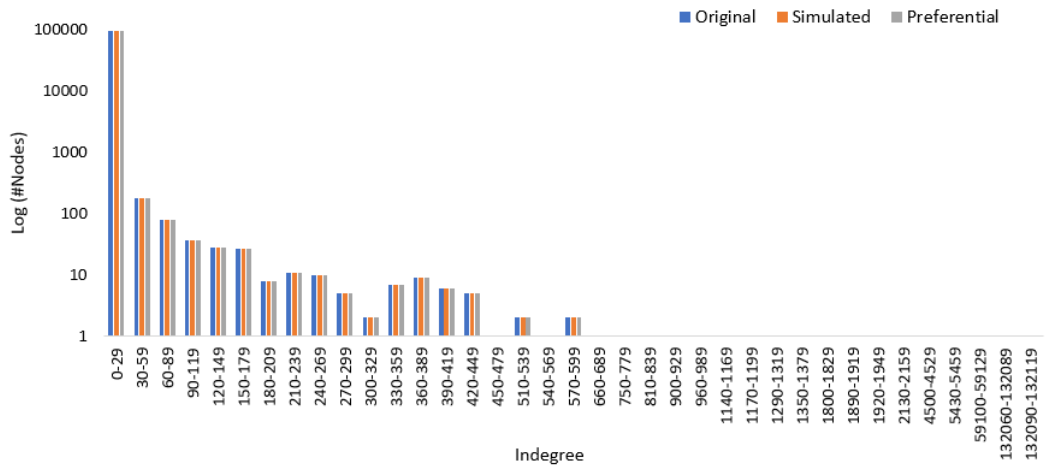


Figure 19: Comparison of indegree histograms for scenario 12.

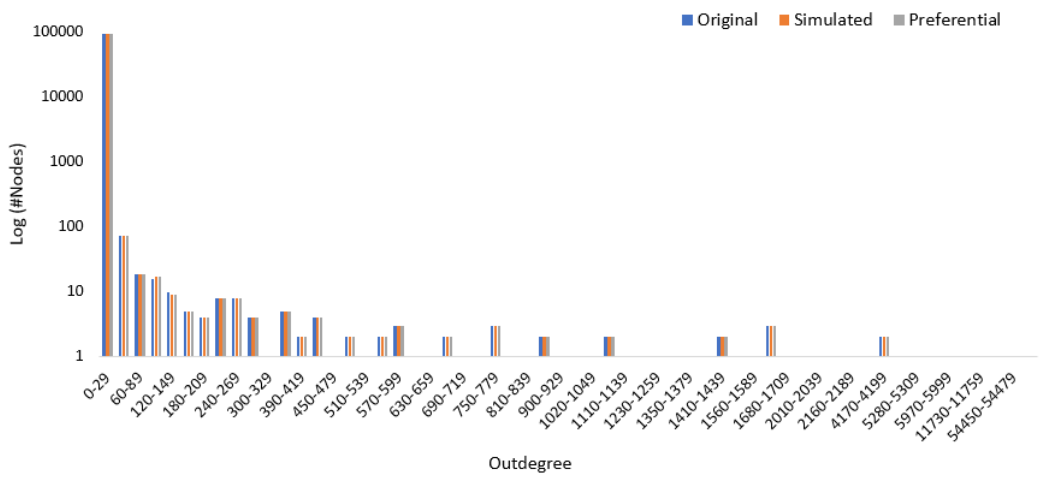


Figure 20: Comparison of outdegree histograms for scenario 12.

4.4 Comparison of the Simulated Scaled-up Graph with the Original Graph

This section describes the results obtained after scaling up the simulated graph 3 times the original graph for scenario 12 of the CTU-13 dataset. We scaled up both of the simulated graphs; one created by our simulation algorithm and another by the PA algorithm. Because of our hardware limitation on the number of available processors, we can scale up the graph only 3 times. But using our Enterprise connection algorithm, graphs can be scaled up linearly with the number of available processors. Here, we compare the botnets subgraphs of the original and the scaled-up simulated graphs (resulting from our algorithm and from the PA algorithm) using the visualization approach. We also compare the histograms of the triangle, indegree, and outdegree distributions of the original graph with the scaled-up simulated graphs.

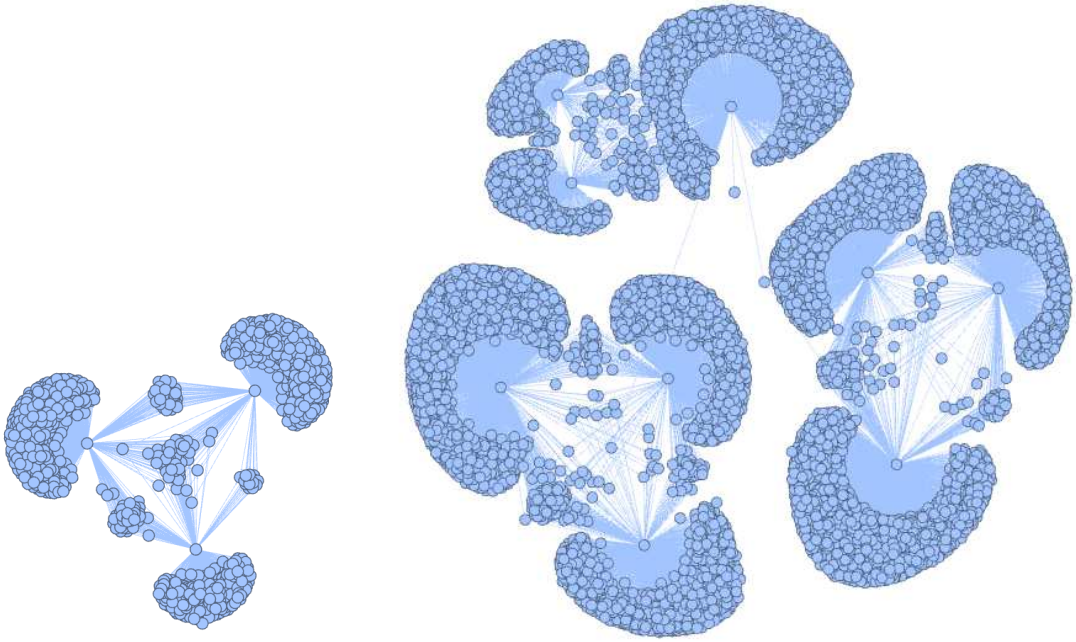
4.4.1 Comparison of Botnet Subgraph

In Figure 21, we compare the scaled-up simulated botnet subgraphs generated by our simulation algorithm and the PA algorithm with the original botnet subgraph using a visualization approach. Similar to Section 4.2.2, we used the *ForceAtlas2* layout of Gephi to draw these figures. The original botnet subgraph used here is the same as the one used in Figure 10a.

In the original botnet subgraph (Figure 21a), there are three major clusters. In our simulated scaled-up botnet subgraph (Figure 21b), there are three subgraphs similar to the original botnet subgraph and each subgraph contains three major clusters similar to the original botnet. Also,

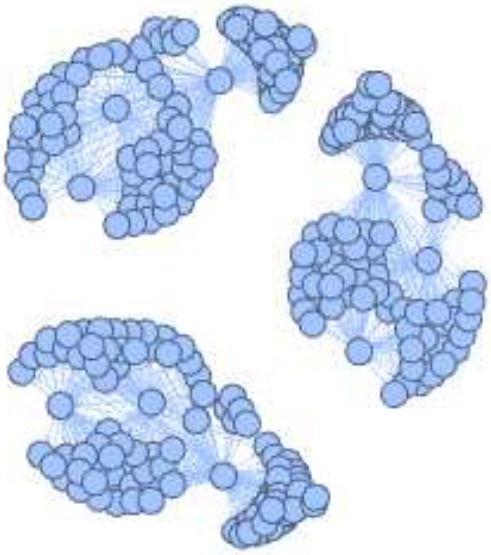
it is very interesting that these three botnet subgraphs are connected with each other. In our simulation approach, we did not connect them intentionally. It happened automatically as a result of how the enterprises got connected with each other. In Enterprise connection algorithm, we choose $N = 10, f = 1/3$, and $P = 3$ that results $E = 20$. We could have chosen one node from each subgraph but as graphs like the CTU-13 contain several nodes with high indegree and outdegree, choosing more nodes introduce more randomness and makes the graph more realistic. These random 20 connections ($E = 20$) among high degree nodes made the botnet subgraphs connected as well.

Figure 21c shows the botnet subgraph simulated by the PA algorithm. There is no visible similarity between the original botnet subgraph and the botnet subgraph simulated by the PA algorithm. Also, from figure 21c, we can see that three botnets subgraphs simulated by each of the processors are not connected to each other. In fact, we had to apply *Gravity* property of Gephi to draw them in close proximity for visualization.



(a) Original botnet.

(b) Scaled-up botnet simulated by our simulation algorithm.



(c) Scaled-up botnet simulated by the PA algorithm.

Figure 21: Comparison of the scaled-up botnets for scenario 12.

4.4.2 Comparison of Overall Graph

Figure 22 shows the comparison of the triangle distribution of the original graph, three times scaled-up graph simulated by our simulation algorithm, and three times scaled-up graph simulated by the PA algorithm. From Figure 22, we see that the PA algorithm generates lots of extra

triangles compared to our simulation methodology. To evaluate how our simulation methodology performed after scaling-up, we plot a separate histogram in Figure 23 to compare our simulated graph with the original graph.

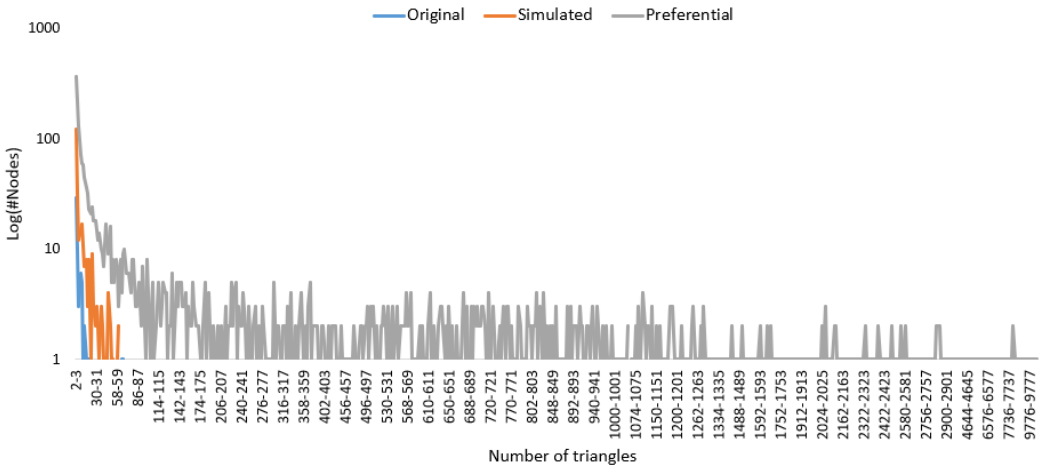


Figure 22: Comparison of triangle histogram of scaled-up graphs for scenario 12 (original vs simulated vs preferential).

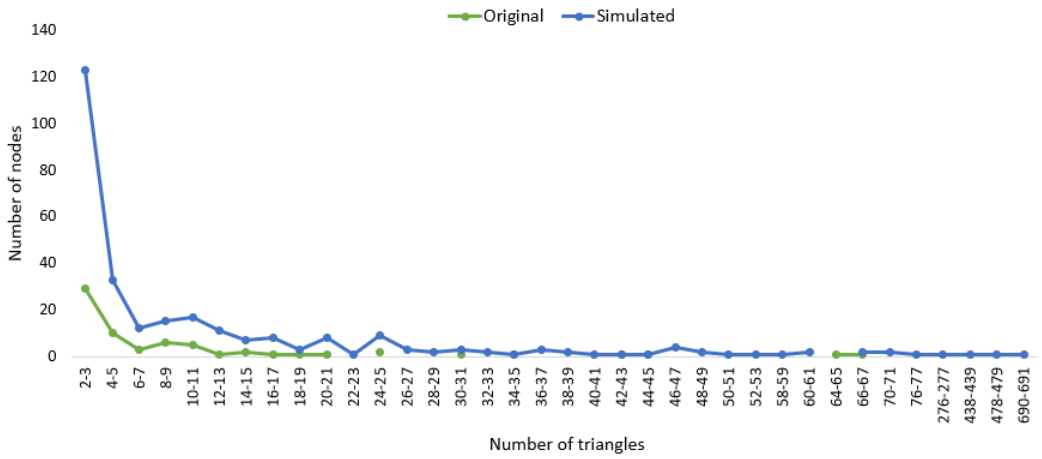


Figure 23: Comparison of triangle histogram between the original and the simulated scaled-up graphs using our algorithm for scenario 12.

As we mentioned earlier in Section 4.3.2 that the number of nodes involved with no and single triangles dominates the histogram, in Figure 23, we show the histogram values for all nodes involved in more than one triangles. In Figure 23, we see that there are some deviations

as well as similarities of the simulated scaled-up graph with the original graph. As the simulated graph is scaled-up to 3 times the original graph, the number of nodes involved with triangles in the simulated graph should also increase approximately 3 times. For the first bin, the number of nodes connected with 2-3 triangles in the simulated graph seems to be 4 times rather than 3 times the number of nodes in the original graph. But for the rest of the bins (4-5, 6-7, 8-9, and the rest of the bins), there is a similar pattern in the histogram and the values are approximately 3 times larger. Besides simulating similar triangle distribution as the original, our simulation algorithm also generates some nodes associated with a large number of triangles.

As we already know from the literature and seen in Section 4.3.2 that the PA algorithm can simulate the same degree distribution as the original graph, it is unnecessary to show any comparison of its performance while simulating the degree distributions. Figures 24 and 25 demonstrate the histograms of indegree and outdegree, respectively, of the original and the scaled-up graphs simulated by our methodology. In both graphs, the number of nodes is represented using a logarithmic scale. For both indegree and outdegree, the simulated large graph has almost the same pattern in the histogram as the original graph, only the number of nodes are scaled up to 3 times. There are small deviations in the simulated scaled-up graph from the original graph in a few instances. For example, in Figure 24, there is one node at the very last bin that is deviated from the original graph.

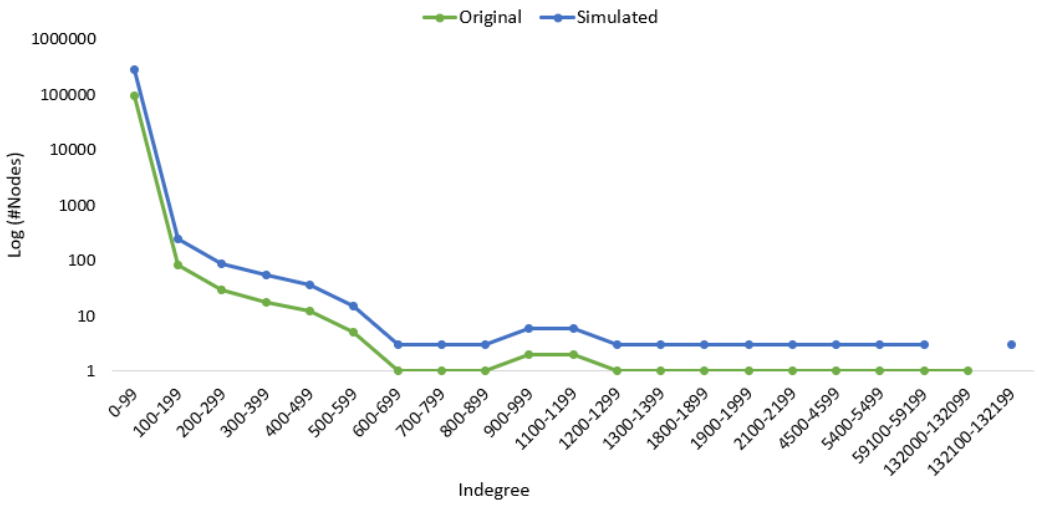


Figure 24: Comparison of indegree histogram between the original and the simulated scaled-up graph for scenario 12.

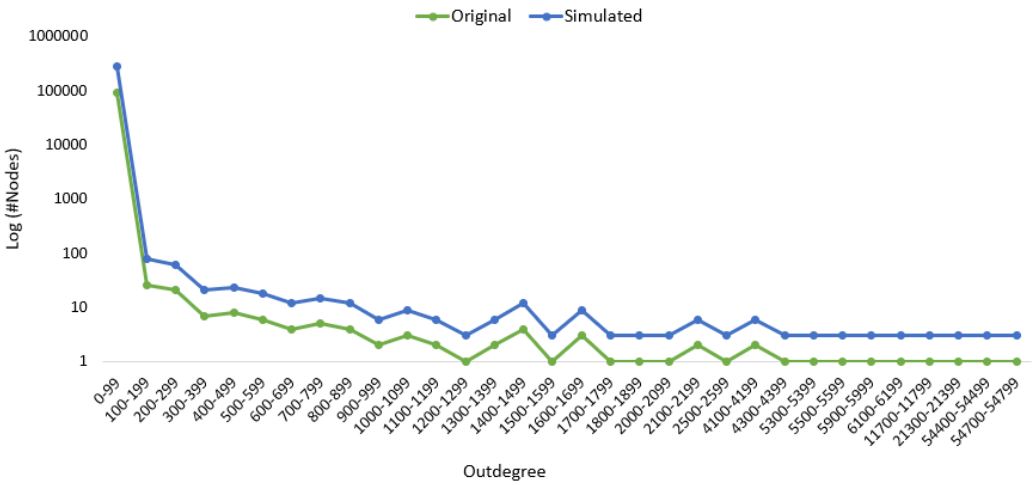


Figure 25: Comparison of outdegree between the original and the simulated scaled-up graph for scenario 12.

The PA algorithm performs well in simulating degree (indegree and outdegree) distributions. But in simulating the triangle distributions, our simulation algorithm outperformed the PA algorithm. Also, in terms of simulating the botnet subgraph, our simulation algorithm performs much better than the PA algorithm.

5 Conclusion and Future Work

In this research, we develop a graph simulation methodology to generate a large-scale, real-life botnet dataset based on the Markov chain and role-mining processes. As one of our main goals is to simulate botnet behavior inside a normal Netflow dataset, we emphasize the botnet subgraph more. This methodology simulates the indegrees, outdegrees, and triangles for each node while hiding the original node identities. Also, we develop an Enterprise connection algorithm that scales up the simulated graph linearly with the number of processors in a distributed system. We compared our simulated graphs with graphs simulated by the PA algorithm and with the given original graph. Our numerical experiments show promising results as the simulated overall graph as well as the simulated botnet traffic is very similar to the original dataset with a small variation. Based on the comparison of histograms, our methodology generates a simulated graph that is almost similar to the original (given) graph. Besides having a better histogram similarity, a similar pattern with small natural randomness is also clearly visible between the simulated and the original botnet subgraph in the visualization of botnet traffic. Also, we see that our simulated scaled-up graph preserves a similar pattern as the original graph based on the number

of triangles, indegrees, outdegrees, and botnets.

We find that the simulation of triangles is a more challenging and complex process compared to the simulation of indegrees and outdegrees. But, from the experimental results, we see that our proposed methodology performed well in simulating the number of triangles. This research can be further extended by improving some limitations of the simulation methodology. In Section 3.6, we observe that while adding the remaining edges, we are adding edges without considering their effect on the existing triangles. As a consequence, the simulation methodology creates undesired triangles in the simulated graph. A possible extension of this research is to modify the methodology to avoid the creation of these undesired triangles.

Declarations

Ethics Approval and Consent to Participate

Not applicable.

Consent for Publication

Not applicable.

Availability of Data and Materials

The data used in this paper is publicly available at: https://www.stratosphereips.org/datasets-ctu13.

Competing Interests

The authors declare that they have no competing interests.

Funding

Not applicable.

Authors' Contributions

THB contributed in conceptualization, implementation, experimentation, and manuscript writing, review, and editing. SH contributed in conceptualization, implementation, experimentation,

and writing the initial manuscript draft. SZ provided guidance, and supervision. HM provided guidance, and supervision.

Acknowledgements

Not applicable.

References

Aiello, W., Chung, F., and Lu, L. (2000). A random graph model for massive graphs. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 171–180. Acm.

Aiello, W., Chung, F., and Lu, L. (2002). Random evolution in massive graphs. In *Handbook of massive data sets*, pages 97–122. Springer.

Albert, R. and Barabási, A.-L. (2000). Topology of evolving networks: local events and universality. *Physical review letters*, 85(24):5234.

Albert, R. and Barabási, A.-L. (2002). Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47.

Amaral, L. A. N., Scala, A., Barthelemy, M., and Stanley, H. E. (2000). Classes of small-world networks. *Proceedings of the national academy of sciences*, 97(21):11149–11152.

Bansal, S., Khandelwal, S., and Meyers, L. A. (2009). Exploring biological network structure with clustered random networks. *BMC bioinformatics*, 10(1):405.

Barabási, A.-L. and Albert, R. (1999). Emergence of scaling in random networks. *science*, 286(5439):509–512.

Becchetti, L., Boldi, P., Castillo, C., and Gionis, A. (2008). Efficient semi-streaming algorithms for local triangle counting in massive graphs. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 16–24. ACM.

Bi, Z., Faloutsos, C., and Korn, F. (2001). The dgx distribution for mining massive, skewed data. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 17–26. ACM.

Bu, T. and Towsley, D. (2002). On distinguishing between internet power law topology generators. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, pages 638–647. IEEE.

Chakrabarti, D. and Faloutsos, C. (2006). Graph mining: Laws, generators, and algorithms. *ACM computing surveys (CSUR)*, 38(1):2.

Chakrabarti, D., Zhan, Y., and Faloutsos, C. (2004). R-mat: A recursive model for graph mining. In *Proceedings of the 2004 SIAM International Conference on Data Mining*, pages 442–446. SIAM.

Chung, F. and Lu, L. (2002a). The average distances in random graphs with given expected degrees. *Proceedings of the National Academy of Sciences*, 99(25):15879–15882.

Chung, F. and Lu, L. (2002b). Connected components in random graphs with given expected degree sequences. *Annals of combinatorics*, 6(2):125–145.

Eckmann, J.-P. and Moses, E. (2002). Curvature of co-links uncovers hidden thematic layers in the world wide web. *Proceedings of the national academy of sciences*, 99(9):5825–5829.

Erdos, P. and Rényi, A. (1960). On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60.

Erdős, P. and Rényi, A. (1961). On the strength of connectedness of a random graph. *Acta Mathematica Hungarica*, 12(1-2):261–267.

Faloutsos, M., Faloutsos, P., and Faloutsos, C. (1999). On power-law relationships of the internet topology. In *ACM SIGCOMM computer communication review*, volume 29, pages 251–262. ACM.

Flake, G. W., Lawrence, S., Giles, C. L., et al. (2000). Efficient identification of web communities. In *KDD*, volume 2000, pages 150–160.

Garcia, S., Grill, M., Stiborek, J., and Zunino, A. (2014). An empirical comparison of botnet detection methods. *computers & security*, 45:100–123.

Gilpin, S., Eliassi-Rad, T., and Davidson, I. (2013). Guided learning for role discovery (glrd): framework, algorithms, and applications. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 113–121. ACM.

Girvan, M. and Newman, M. E. (2002). Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826.

Gleeson, J. P. (2009). Bond percolation on a class of clustered random networks. *Physical Review E*, 80(3):036107.

Guo, W. and Kraines, S. B. (2009). A random network generator with finely tunable clustering coefficient for small-world social networks. In *Computational Aspects of Social Networks, 2009. CASON'09. International Conference on*, pages 10–17. IEEE.

Harun, S., Bhuiyan, T. H., Zhang, S., Medal, H., and Bian, L. (2017). Bot classification for real-life highly class-imbalanced dataset. In *Dependable, Autonomic and Secure Computing, 15th Intl Conf on Pervasive Intelligence & Computing, 3rd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech), 2017 IEEE 15th Intl*, pages 565–572. IEEE.

Henderson, K., Gallagher, B., Eliassi-Rad, T., Tong, H., Basu, S., Akoglu, L., Koutra, D., Faloutsos, C., and Li, L. (2012). Rolx: structural role extraction & mining in large graphs. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1231–1239. ACM.

Klauck, H., Nanongkai, D., Pandurangan, G., and Robinson, P. (2015). Distributed computation of large-scale graph problems. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 391–410. Society for Industrial and Applied Mathematics.

Kleinberg, J. M., Kumar, R., Raghavan, P., Rajagopalan, S., and Tomkins, A. S. (1999). The web as a graph: measurements, models, and methods. In *International Computing and Combinatorics Conference*, pages 1–17. Springer.

Kohli, R. and Sah, R. K. (2003). Market shares: Some power law results and observations.

Kotenko, I., Konovalov, A., and Shorov, A. (2010). Agent-based modeling and simulation of botnets and botnet defense. In *Conference on Cyber Conflict. CCD COE Publications. Tallinn, Estonia*, pages 21–44.

Kumar, R., Raghavan, P., Rajagopalan, S., and Tomkins, A. (1999). Extracting large-scale knowledge bases from the web. In *VLDB*, volume 99, pages 639–650.

Leskovec, J., Chakrabarti, D., Kleinberg, J., and Faloutsos, C. (2005a). Realistic, mathematically tractable graph generation and evolution, using kronecker multiplication. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 133–145. Springer.

Leskovec, J., Chakrabarti, D., Kleinberg, J., Faloutsos, C., and Ghahramani, Z. (2010). Kronecker graphs: An approach to modeling networks. *Journal of Machine Learning Research*, 11(Feb):985–1042.

Leskovec, J. and Faloutsos, C. (2007). Scalable modeling of real graphs using kronecker multiplication. In *Proceedings of the 24th international conference on Machine learning*, pages 497–504. ACM.

Leskovec, J., Kleinberg, J., and Faloutsos, C. (2005b). Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD*

international conference on Knowledge discovery in data mining, pages 177–187. ACM.

Leskovec, J., Rajaraman, A., and Ullman, J. D. (2014). *Mining of massive datasets*. Cambridge university press.

Medina, A., Matta, I., and Byers, J. (2000). On the origin of power laws in internet topologies. *ACM SIGCOMM computer communication review*, 30(2):18–28.

Moody, J. (2001). Race, school integration, and friendship segregation in america. *American journal of Sociology*, 107(3):679–716.

Murphy, R. C., Wheeler, K. B., Barrett, B. W., and Ang, J. A. (2010). Introducing the graph 500. *Cray User's Group (CUG)*, 19:45–74.

Neukum, G. and Ivanov, B. (1994). Crater size distributions and impact probabilities on earth from lunar, terrestrial-planet, and asteroid cratering data. *Hazards due to Comets and Asteroids*, 359.

Newman, M. E. (2005). Power laws, pareto distributions and zipf's law. *Contemporary physics*, 46(5):323–351.

Newman, M. E. (2009). Random graphs with clustering. *Physical review letters*, 103(5):058701.

Newman, M. E., Watts, D. J., and Strogatz, S. H. (2002). Random graph models of social networks. *Proceedings of the National Academy of Sciences*, 99(suppl 1):2566–2572.

Pennock, D. M., Flake, G. W., Lawrence, S., Glover, E. J., and Giles, C. L. (2002). Winners don't take all: Characterizing the competition for links on the web. *Proceedings of the national academy of sciences*, 99(8):5207–5211.

Redner, S. (1998). How popular is your paper? an empirical study of the citation distribution. *The European Physical Journal B-Condensed Matter and Complex Systems*, 4(2):131–134.

Rossi, R. A. and Ahmed, N. K. (2015). Role discovery in networks. *IEEE Transactions on Knowledge and Data Engineering*, 27(4):1112–1131.

Rossi, R. A., Gallagher, B., Neville, J., and Henderson, K. (2013). Modeling dynamic behavior in large evolving graphs. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 667–676. ACM.

Rossow, C., Dietrich, C. J., Grier, C., Kreibich, C., Paxson, V., Pohlmann, N., Bos, H., and Van Steen, M. (2012). Prudent practices for designing malware experiments: Status quo and outlook. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 65–79. IEEE.

Ruan, Y. and Parthasarathy, S. (2014). Simultaneous detection of communities and roles from large networks. In *Proceedings of the second ACM conference on Online social networks*, pages

203–214. ACM.

Sala, A., Cao, L., Wilson, C., Zablit, R., Zheng, H., and Zhao, B. Y. (2010). Measurement-calibrated graph models for social network experiments. In *Proceedings of the 19th international conference on World wide web*, pages 861–870. ACM.

Sarma, A. D., Holzer, S., Kor, L., Korman, A., Nanongkai, D., Pandurangan, G., Peleg, D., and Wattenhofer, R. (2012). Distributed verification and hardness of distributed approximation. *SIAM Journal on Computing*, 41(5):1235–1265.

Seshadhri, C., Kolda, T. G., and Pinar, A. (2012). Community structure and scale-free collections of erdős-rényi graphs. *Physical Review E*, 85(5):056109.

Shiravi, A., Shiravi, H., Tavallaee, M., and Ghorbani, A. A. (2012). Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *computers & security*, 31(3):357–374.

Siddharthan, A. (2002). Christopher d. manning and hinrich schutze. foundations of statistical natural language processing. mit press, 2000. isbn 0-262-13360-1. 620 pp. $64.95/£ 44.95 (cloth). *Natural Language Engineering*, 8(1):91–92.

Somaiya, M., Jermaine, C., and Ranka, S. (2010). Mixture models for learning low-dimensional roles in high-dimensional data. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 909–918. ACM.

Tauro, S. L., Palmer, C., Siganos, G., and Faloutsos, M. (2001). A simple conceptual model for the internet topology. In *Global Telecommunications Conference, 2001. GLOBECOM'01. IEEE*, volume 3, pages 1667–1671. IEEE.

Travers, J. and Milgram, S. (1967). The small world problem. *Phychology Today*, 1(1):61–67.

Vignau, B., Khoury, R., Hallé, S., and Hamou-Lladj, A. (2021). The botnet simulator: A simulation tool for understanding the interaction between botnets. *Software Impacts*, 10:100173.

Watts, D. J. and Strogatz, S. H. (1998). Collective dynamics of 'small-world'networks. *nature*, 393(6684):440.

Waxman, B. M. (1988). Routing of multipoint connections. *IEEE journal on selected areas in communications*, 6(9):1617–1622.

Willis, J. C. and Yule, G. U. (1922). Some statistics of evolution and geographical distribution in plants and animals, and their significance.

Zhang, F., Zhang, S., Lightsey, C., Harun, S., and Wong, P. C. (2018). Bgs: A large-scale graph visualization tool. *Electronic Imaging*, 2018(1):1–9.