A Comparison of Graph-Based Synthetic Data Generators for Benchmarking Next-Generation Intrusion Detection Systems

Stefano Iannucci, Hisham A. Kholidy,
Amrita Dhakal Ghimire, Rui Jia, Sherif Abdelwahed
Distributed Analytics and Security Institute
Mississippi State University

Ioana Banicescu
Department of Computer Science and Engineering
Mississippi State University

***Abstract*—Property-graphs are becoming popular for Intrusion Detection Systems (IDSs) because they allow to leverage distributed graph processing platforms in order to identify malicious network traffic patterns. However, a benchmark for studying their performance when operating on big data has not yet been reported. In general, benchmarking a system involves the execution of workloads on datasets, where both of them must be representative of the application of interest. However, few datasets containing real network traffic are openly available due to privacy concerns, which in turn could limit the scope and results of the benchmark. In this work, we build two synthetic data generators for benchmarking next generation IDSs by introducing the support for property-graphs in two well-known graph generation algorithms: Barabási-Albert and Kronecker. We run an extensive experimental evaluation using a publicly available dataset as seed for the data generation, and we show that the proposed approach is able to generate synthetic datasets with high veracity, while also exhibiting linear performance scalability.**

I. Introduction

Network intrusion detection has been an active area of research for the past two decades due to the significant impact on social and economic activities, as almost every aspect of modern day life is now dependent on networked systems. Presently, several publicly available network Intrusion Detection Systems (IDSs) (e.g., Bro [1], Snort [2], ACARM-ng [3], AIDE [4], etc.), are either unable to detect coordinated attacks, or provide false positives at a rate that might not be acceptable [5]. To overcome these limitations, the graph-based approaches initially proposed in [5], [6], [7], [8], [9], [10] are now becoming popular for intrusion detection, as they can be used to model either the network traffic, the attacker behavior or the defended system. Furthermore, graph-based approaches allow to leverage distributed graph processing platforms for the identification of malicious network traffic patterns. However, especially when graphs are used to model network traces, they can reach sizes that make them difficult, and even impossible to be analyzed with a single host. Thus, the intrusion detection problem quickly becomes a big data problem.

Several big data processing tools are currently available (e.g., Apache Hadoop [11], Apache Spark [12], Apache Hive [13] and Shark [14] among the others [15]), and their performance can be measured with big data benchmarks (e.g., [16], [17]). A big data benchmarking system evaluates a big data environment based on four properties: volume, velocity, variety, and veracity. The *Volume* measures the amount of data a big data system can handle. The *Velocity* measures the maximum rate at which the data can be analyzed in a streaming system, or the maximum update rate of the datasets in case of non-streaming systems. The *Variety* measures the flexibility of tools and platforms in handling heterogeneous data. The *Veracity* reflects the closeness of the synthetic data compared to its counterpart, which is usually represented by either real-world data, or by the seed data used for generating the synthetic data. Each benchmark contains specific datasets and workloads for specific application domains (e.g., web search, e-commerce), and the results obtained with a certain benchmark cannot be generalized to other application domains because of the datasets, workloads and performance metrics of interest used. In general, benchmarking a system involves the execution of workloads on datasets, where both of them must be representative of the application of interest. However, due to privacy concerns, few datasets containing real network traffic are openly available, and that could limit the scope and results of the benchmark. To be representative from the workload perspective, the benchmark must include typical operations executed in the cyber-security domain, such as queries on nodes, edges, paths, and sub-graphs.

On one hand, in order to limit the damage produced by the attacker to the target system [18], having a clear idea of the performance, in terms of threat detection time, and of the scalability of a graph-based IDS is paramount. On the other hand, in spite of their practical applications success, it is difficult to make deployment decisions among the large variety of big data processing platforms because of the lack of comprehensive understanding of their performance in a cyber-security setting. In other words, a specific cyber-security application benchmark is necessary before system designers, programmers, and researchers within the domain of cyber-security can optimize the performance, resilience, and energy efficiency of these systems.

The main contribution of this paper aims to address the

2168-9253/17 $31.00 © 2017 IEEE
DOI 10.1109/CLUSTER.2017.54

lack of openly available data sources by introducing two novel synthetic data generation models that are able to capture all the features of a network trace (e.g., duration of the connections, amount of transferred data, amount of transferred packets). To this end, we extend two well-known graph generation models, based on the Barabási-Albert (BA) [19] and Kronecker [20] algorithms, to support property-graphs [21] for the representation of Netflow [22] data. Both models work by growing an existing seed graph to a graph of arbitrary size, where the seed can be any network trace provided in the PCAP format [23].

Even though several graph databases supporting property-graphs exist (e.g., Neo4J [24], Titan [25], DEX [26]), they are usually suited for applications requiring low-latency and high-throughput data analysis [27], such as recommendation systems. Therefore, they are focusing on the parallelization of queries streams, rather than on the parallelization of a single query execution. For this reason, as a second contribution, we implement two data generation algorithms on the only distributed graph processing platform available at this time that supports property graphs: Apache Spark with the GraphX library [28]. The latter provides a Map-Reduce programming framework [29] that can be leveraged to parallelize the execution of the data generation by taking advantage of a cluster of compute nodes.

The third contribution of the paper is an extensive experimental evaluation of the proposed models, in order to show that the synthetically generated data exhibits high veracity, that is, a high degree of similarity in terms of degree distributions and PageRank [30] distributions with respect to the seed data. Furthermore, we show that our approach is able to generate big datasets containing billions of edges in less than a hour when distributing the load on 60 compute nodes. Finally, we empirically prove that the proposed algorithms have a linear scalability and that the execution platform scales linearly according to the number of compute nodes used for the generation. As a last contribution, and in order to ease the reproducibility of the experiments, we release with the open-source GPLv3 license the source code of the application, and make it available on GitHub at: https://github.com/msstate-dasi/csb

The paper is organized as follows. In Section II we introduce related work in graph generation models. In Section III, we provide an in-depth description of the extension to the Kronecker and BA algorithms. In Section IV, we present an anomaly detection approach based on property-graphs. The experimental evaluation is described in Section V, where we discuss in detail the performance and veracity aspects of the data generation. Finally, in Section VI, we draw conclusions and discuss future work.

II. Background and Related Work

Complex systems and their properties are sustained in nature and are object of research in many areas of science, society and technology. These systems consist of a large or extremely large number of components that interact via networks. Example of such systems are: social networks [31], [32], biological networks [33], the Internet [34], and others. Historically, complex systems and networks have often been modeled as random graphs for the purpose of studying their behaviors.

The interconnectivity in networks is fundamental to the behavior of complex systems. In recent years tools and data became available to probe their topology, and that was essential for the understanding of the role and impact of the underlying connectivity on system's behavior. It has been concluded that, without exploiting the network topology of a complex system, its behavior cannot be understood or predicted [35].

Over the years, a number of real world systems and networks have been modeled and analyzed using random graph models. In complex systems using random graph models, relevant properties, such as degree distributions and clustering coefficients, can be incorporated. Degree distribution is one of the most prominent features of these networks. Models such as Erdős-Rényi (ER) [36], [37], stochastic block model (SBM) [38], Barabási-Albert (BA) [19], exponential random graph [39], [40], recursive matrix [41], stochastic Kronecker (SKG) [42], [20], have been developed to capture the diversity of the degree distributions. In their development, each of these models consider some aspects of the networks.

Over half a century ago, the Erdős-Rényi (ER) model [36] was introduced, drawing attention to interconnected systems. The model, which after being introduced was widely used in biology, sociology and computer science, assumed that complex systems are wired randomly together. Almost two decades ago, Watts-Strogatz (WS) model [43] was proposed, in which vertices form a one-dimensional lattice where each vertex is connected to its two nearest and next-nearest neighbors. This process of generating long-range connections determined a decrease in the distance between vertices leading to a small-world phenomenon. However, it has been found that a common feature which both ER and WS models exhibit is that the probability of finding a highly connected vertex (with a high degree value) decreases exponentially with the degree value of the vertex (resulting in a small or zero number of highly connected vertices). This research result was found to be completely opposed to the phenomena observed in network studies, where highly connected vertices have large chance of occurring. To study the community structures found in many real-world systems, the stochastic block model (SBM) was proposed [38].

Barabási and Albert [19] point out two generic aspects of real networks that have been missed being incorporated in the ER and WS models above. The two generic aspects have been shown to be in contradiction with properties observed in large networks [19]. The first generic aspect emphasizes that both models consider starting with a fixed number of vertices that are randomly connected (ER model), or reconnected (WS model), without changing the number of vertices. In reality, most networks are open, and grow continuously by adding new vertices to the system. In this way, the number of vertices increases continuously throughout the lifetime of the network. The second generic aspect shows that in both models, the probability of two vertices being connected is random and uniform. However, in reality, most networks exhibit preferen-

tial connectivity (preferential attachment), indicating that the probability with which a new vertex connects to the existing vertices is not uniform. It has been shown that there is a higher probability that a new vertex will be connected to a vertex that already has a large number of connections. The authors show that a common property exhibited by many large networks is that the vertex connectivity follows a scale-free power law distribution. Specifically, they show that large networks self-organize into a scale-free state following a power law distribution independently of the system and the identity of its constituents. The essence of the Barabási-Albert (BA) model [35] is that the probability $P(k)$ that a vertex in the network interacts with k other vertices decays as a power law, following $p(x) \propto L(x)x^{-\alpha}$, where $\alpha > 1$ is the power-law exponent and $L(x)$ is a slowly varying function, that is, it respects $\lim_{x\to\infty} \frac{L(tx)}{L(x)} = 1$ with constant t.

In recent years, it became obvious that no networks seen in nature, science, economics, or technology are completely random. Their evolution is shaped by mechanisms beyond randomness. However, the models mentioned above generate graphs with a specific type of degree distribution. It has been shown that various phenomena can be studied and predictions can be inferred by the use of the universality of topological characteristics, such as degree distributions, degree correlations, communities, and others [35].

During the last decade, new models have been developed. The Chung-Lu (CL) model [44], [45] can generate a random graph with a given sequence of expected degrees, and is capable of generating networks from almost any real-world desired degree distribution. The block two-level Erdős-Rényi (BTER) model [46], [47] has recently been developed for the study of the community structure by capturing the degree distribution and clustering coefficients.

The continuous need to better understand the behavior of complex systems and networks drives the requirement of generating larger and larger networks, which in turn requires the efficient generation of massive random networks or graphs. It has been shown that analyzing a large complex system or network by generating it using a small model of it, doesn't produce accurate results. The small network may not exhibit the same complex collective behavior that is exhibited by the large network, even if both networks have been generated using the same model [48]. Recently, a Degree Grouping (DG) method for developing time and space efficient sequential and distributed-memory parallel algorithms that generate random graphs for the CL, SBM and BTER models has been proposed, and its effectiveness has been demonstrated [49]. Moreover, distributed-memory parallel implementations for the SKG and the BA models have also been reported in the literature ([50], [51], [20]).

There are many examples of Next-Generation IDSs based on graphs (e.g., [52], [53], [54], [55], [56], [57]), and several attempts were made to use the NetFlow protocol ([58], [59], [60], [61]) to detect DoS attacks such as Smurf, and worms such as W32.Blaster Worm and Red Worm. However, none of them are able to leverage the modeling of Netflow-based data using property-graphs. Leveraging property-graphs capabilities to efficiently analyze network data is a recent trend that has been initially proposed by SANS [62] and recently extended by Oracle [63]. On one hand, from a cyber-security perspective, it is important for an IDS to guarantee a timely identification of potential threats, in order to limit the negative impact that they could have on the system [18]. On the other hand, none of the existing works aim at evaluating the performance, in terms of execution time and memory usage, of next-generation IDSs based on graphs. This work tries to fill this gap by introducing a vital component of a benchmark for property-graph based IDSs: a scale-free synthetic data generator of property-graphs representing real network traffic.

III. Data Generators

In this section, we describe the sequence of steps undertaken for the generation of the synthetic datasets representing network traffic based on directed property-graphs. A property-graph is formalized as $G = (V, E, D_v, D_e)$, where V is the set of vertices, E is the *multi*-set of edges, D_v is the set of data associated with each vertex and D_e is the set of data associated with each edge. The property-graphs (i) allow for the existence of multiple edges between any couple of nodes, and (ii) support the association of data structures containing multiple attributes with vertices and edges. In our work, we use property-graphs to model Netflow network data. Among other formats, we consider Netflow because we aim at building the data generator component for a benchmark for future generation IDSs, and we want to use a data format that can be used as an evidence in the court [64]. Specifically, we represent Netflow data in a property-graph format as follows: we map on V the sets of the hosts, and on E the `TCP` connections or `UDP` streams between any couple of hosts. We only consider a single attribute for D_v, that is, *ID*, which univocally identifies the vertices in the graph. Instead, we consider the following Netflow attributes for D_e:

- *PROTOCOL.* Represents the transport protocol of the data stream. Currently supported protocols are `TCP` and `UDP`.
- *SRC_PORT.* Represents the source port of the data stream.
- *DEST_PORT.* Represents the destination port of the data stream.
- *DURATION.* Represents the duration of the data stream, expressed in milliseconds.
- *OUT_BYTES.* Represents the amount of data transferred (in bytes) from source to destination.
- *IN_BYTES.* Represents the amount of data transferred (in bytes) from destination to source.
- *OUT_PKTS.* Represents the amount of packets transmitted from source to destination.
- *IN_PKTS.* Represents the amount of packets transmitted from destination to source.
- *STATE.* This attribute is used only in the case the edge represents a TCP connection and contains its state.

We introduce the support for property-graphs in two well known data generation algorithms, namely Barabási-Albert

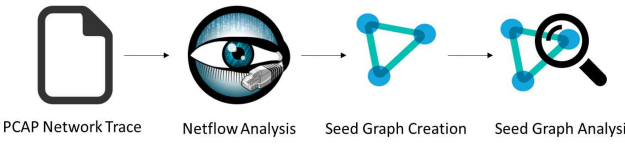

Fig. 1. Preliminary Steps

(BA) [65] and Kronecker [20]. Both of these algorithms work by expanding a *seed graph* while trying to maintain its structural properties and attributes' properties in a scale-free fashion. In this work, we focus on the *in-degree*, *out-degree* [66] and *PageRank* [67] structural properties because they can be obtained even for relatively large graphs [68]. However, the modular architecture of our software prototype and the framework on top of which is built can easily support additional generation methods that can take into account more properties, such as the *betweenness centrality* [69] and *connected components* [70].

The creation of the seed graph is accomplished according to the steps shown in Figure 1: the process starts with some source data in PCAP format and its Netflow representation is obtained by analyzing it with Bro IDS [71]. Netflow data is then mapped to a property-graph and finally, an analysis of the structural and attributes' properties is carried out. The last step is aimed at analyzing the structural characteristics and the properties of the seed graph. Specifically, we compute the in- and out-degree probability distributions that will be used later, in the data generation phase, to tune the power-law distribution, in order to ensure that the original structural characteristics are maintained in the synthetic graph. Furthermore, we also compute the probability distributions of all the seed graph's properties. To this end, and in order to generate meaningful Netflow attributes, we first compute $p(IN\_BYTES)$, that is, the unconditional probability distribution of the IN_BYTES attribute; then, for all the other Netflow attributes a, we compute $p(a|IN\_BYTES)$.

A. Data Generator based on Barabási-Albert

The BA model is extensively discussed in [65]. Since most of the real complex networks have a degree distribution that follows the power law, the authors leverage the concept of preferential attachment, such that the likelihood of connecting to a node depends on the node's degree. The BA graph generation algorithm is iterative and each iteration is composed of two steps: *growth* and *preferential attachment*. During the first step, a new vertex v is added to the existing graph composed of m_0 vertices, and $m \leq m_0$ edges are created with v as the source and no destination. The second step involves the selection of edges' destinations, which are chosen according to the following probability distribution: $\pi(i) = \frac{k_i}{\sum_j k_j}$, where i is the index of the destination vertex and k_i is its in-degree. One of the main limitations of this algorithm resides in its scalability, because $\pi(i)$ must be updated at every iteration and because only a single vertex is added at every iteration. A more

Input: $Graph\ G = (V, E, D_v, D_e)$
Input: $long\ desired\_size$
Input: $double fraction$
Input: $Distribution\ outDegree$
Input: $Distribution\ inDegree$
Input: $Distribution[]\ properties$
Output: $Graph\ G\prime = (V\prime, E\prime, D_v\prime, D_e\prime)$
1: $G\prime = (V\prime, E\prime, D_v\prime, D_e\prime) \leftarrow G$
2: **while** $(|E\prime| < desired\_size)$ **do**
3: $Edge[]\ sampledEdges \leftarrow sample(E\prime, fraction)$
4: $Vertex[]\ newVertices \leftarrow createEmptyVertices(|sampledEdges|)$
5: $V\prime \leftarrow V\prime \cup newVertices$
6: **for** $(i = 0\ to\ |sampledEdges|)$ **do**
7: $Vertex\ destVertex \leftarrow random(sampledEdges[i])$
8: $long\ inDegree \leftarrow sample(inDegree)$
9: $long\ outDegree \leftarrow sample(outDegree)$
10: $Edge[]\ outEdges \leftarrow createOutEdges(outEdges, newVertices[i], destVertex)$
11: $Edge[]inEdges \leftarrow createInEdges(inEdges, destVertex, newVertices[i])$
12: $E\prime \leftarrow E\prime \cup outEdges \cup inEdges$
13: **end for**
14: **end while**
15: $SetD_e\prime \leftarrow \emptyset$
16: **for** $Edge\ e \in E\prime$ **do**
17: **for** $Distribution\ d \in properties$ **do**
18: $D_e\prime \leftarrow D_e\prime \cup addProperty(e, sample(d))$
19: **end for**
20: **end for**
21: **return** $Graph\ G\prime = (V\prime, E\prime, D_v\prime, D_e\prime)$

Fig. 2. Extended Barabási-Albert Algorithm

recent work [50] discusses a parallel extension of BA, where an edge list data structure is exploited in order to avoid the re-computation of $\pi(i)$, and a constant number of vertices is added at each iteration. The preferential attachment is realized as a two-stages process, by sampling an edge from the edge list with a uniform probability distribution, and then by randomly selecting one of the connected vertices. The rationale is that the number of occurrences of a certain vertex in an edge list is equal to its degree, and this approach is able to add an edge in constant time.

We propose a variant of [50], hereafter referred to as Property-Graph Parallel Barabási-Albert (PGPBA), which introduces the support for property-graphs, as detailed in Figure 2. Differently from the former algorithm, which adds a constant number of vertices at each iteration, we support a fixed granularity level by keeping constant the ratio between the amount of newly added vertices and the amount of edges in the graph, defined as $fraction = \frac{newVertices}{|E\prime|}$.

The inputs of PGPBA are: the seed graph G, the desired size of the synthetic graph measured in edges, the fraction of nodes to add with respect to the current size of the synthetic graph, the pre-computed in- and out-degree distributions, and the distributions of the edges' properties. The main algorithm is enclosed in the `while` loop (lines 2-14), which iterates until the output graph has the required number of edges. Line 3 implements the first stage of the preferential attachment, where

the edges list is sampled to retrieve $fraction*E\prime$ edges. A new vertex is created (line 4), added to the synthetic graph (line 5), and then associated to one of the vertices contained in the sampled edges (line 7). On lines 8 and 9, we sample the out- and in-degree distributions. The sampled values will establish, respectively, how many edges must connect the new vertex to its associated one, and vice-versa. The actual edges are created on line 10 and 11, and added to the synthetic graph on line 12. All these steps are executed in $O(|E\prime|)$. Finally, we generate $D_e\prime$ by sampling the pre-computed Netflow attributes' distributions (line 15-20) in $O(|E\prime| \times |properties|)$, and the synthetic graph $G\prime$ is returned on line 21. The overall complexity of the presented algorithm is therefore $O(|E\prime| \times |properties|)$. In a Spark Map-Reduce perspective, PGPBA is implemented by using the `RDD.sample()` function on the edges' RDD (Resilient Distributed Dataset) to extract a subset of the edges of the current graph. The latter is then equally partitioned among the available compute nodes and, for every edge, a new vertex is created and attached as its source. The edges' destination vertex instead remains the same to simulate the attachment to existing vertices.

B. Data Generator based on Kronecker

The Kronecker model for synthetic graphs generation is extensively discussed in [20]. The authors propose two versions of the data generation algorithm. The first version, often referred to as the *deterministic* Kronecker, is based on the concept of Kronecker matrix multiplication, which is used to iteratively multiply a deterministic adjacency matrix by itself until the number of generated vertices V is greater or equal to the desired size of the synthetic graph. The complexity of the deterministic Kronecker is $O(|V|^2)$. The second version of the algorithm, often referred to as the *stochastic* Kronecker, is instead based on stochastic adjacency matrices. Each entry $\theta_{i,j}$ of a stochastic adjacency matrix represents the probability value that an edge can appear between vertex i and vertex j, and $\sum \theta_{i,j} = |E|$, where $|E|$ is the expected number of edges given by the realization of the matrix. Instead of directly executing the Kronecker multiplication, which has $(O|V|^2)$ complexity, the idea behind the stochastic Kronecker is to simulate the behavior of the deterministic version by randomly generating an amount of edges equal to the number of edges expected for the $k-th$ iteration of the deterministic multiplication, that is, $|E|^k$. The whole algorithm is divided into two main steps: (i) the generation of the initiator matrix using the *Kronfit* [42] fitting procedure, and (ii) the generation of the synthetic data by placing $|E|^k = |E\prime|$ edges by directly simulating the Kronecker product. The complexity of stochastic Kronecker is $O(|E\prime|)$.

We propose a variant of the stochastic Kronecker, hereafter referred to as the Property-Graph Stochastic Kronecker (PGSK), which adds to it the features needed to support property-graphs. The PGSK algorithm is illustrated in Figure 3. Its inputs are a seed property-graph G, the desired size of the synthetic graph, the pre-computed out-degree distribution, and the distributions of the edges' properties. The graph

Input: $Graph\ G = (V, E, D_v, D_e)$
Input: $long\ desired\_size$
Input: $Distribution\ outDegree$
Input: $Distribution[]\ properties$
Output: $Graph\ G\prime = (V\prime, E\prime, D_v\prime, D_e\prime)$
1: $SetE^p \leftarrow \emptyset$
2: **for** $Edgee \in E$ **do**
3: $E^p \leftarrow E^p \cup e$
4: **end for**
5: $Graph\ G^p \leftarrow (V, E^p)$
6: $Graph\ G^r \leftarrow Kronfit(G^p)$
7: $Graph\ G^k = (V\prime, E^k) \leftarrow Kronecker(G^r, desired\_size)$
8: $SetE\prime \leftarrow \emptyset$
9: **for** $Edgee \in E^k$ **do**
10: $long\ n \leftarrow sample(outDegree)$
11: $E\prime \leftarrow E\prime \cup createEdge(e.src, e.dst, n)$
12: **end for**
13: $SetD_e\prime \leftarrow \emptyset$
14: **for** $Edge\ e \in E\prime$ **do**
15: **for** $Distribution\ d \in properties$ **do**
16: $D_e\prime \leftarrow D_e\prime \cup addProperty(e, sample(d))$
17: **end for**
18: **end for**
19: **return** $Graph\ G\prime = (V\prime, E\prime, D_v\prime, D_e\prime)$

Fig. 3. Property-Graph Stochastic Kronecker (PGSK) Algorithm

G is used as a seed, that is, as a template providing structural properties and attributes that must be reflected in the generated synthetic graph. Lines 1-5 are used to obtain a standard graph representation G^p of the input property-graph G, where only a single edge is kept between any couple of vertices, if at least one edge was present between the same two vertices in G and all the vertices and edges attributes are removed. This can be effectively accomplished in $O(|E|)$ by converting the multi-set E in the set E^p using a hashed data structure. Although this step seems to be memory inefficient at first sight, we note that $|E^p| \leq |E|$, therefore the peak memory needed to perform it is at most $O(2 \times |E|)$, which is order of magnitudes lower than the memory required for the synthetic graph. The graph G^p has a format compatible with that of the stochastic Kronecker. Lines 6-7 in Figure 3 are used to execute respectively Kronfit and the Kronecker expansion in $O(|E\prime|)$. The latter is a Map-Reduce parallel implementation of the recursive descent described in [20], where an initially empty RDD of edges, with size equal to the number of edges to generate, is partitioned among the available compute nodes. These compute nodes independently generate edges that might conflict because of the probabilistic descent, therefore, at the end of the generation, a `RDD.distinct()` function is invoked on the graph's edges RDD to ensure that only distinct edges are kept. The parallel implementation of the recursive descent is called until the number of generated edges is equal to or greater to the number of expected edges given by the probabilistic initiator matrix. Subsequently, we duplicate every edge of the resulting graph G^k according to the distribution of the outgoing edges (lines 9-12) in $O(|E\prime|)$, and we generate $D_e\prime$ by sampling the pre-computed Netflow attributes' distributions (line 13-18) in $O(|E\prime| \times |properties|)$. Finally, the synthetic graph $G\prime$ is

returned on line 19. The overall complexity of the presented algorithm is $O(|E\prime| \times |properties|)$.

IV. A Netflow Based Anomaly Detection Approach

In this section, we introduce a first attempt to use the Netflow-based graphs data for intrusion detection purposes. Some attacks have traffic patterns that cannot be characterized by only one flow, and instead require an aggregation of the related flows' information. Property-graphs can improve the performance in the processing of aggregated packet data because they provide a powerful data structure that can be leveraged to efficiently aggregate different connection's information. The performance comparison between currently existing IDSs and property-graph based IDSs is out of the scope of this paper and it will be addressed in a future work, as described in Section VI.

The proposed detection approach extends some works in [72] and [73], and it is mainly based on the examination of the parameters of traffic patterns to discover traffic used in attacks through any changes in the payload or port numbers. Using this approach, we can detect several attacks based on network traffic such as flooding and scanning attacks, including Denial of Service (DoS) and Distributed Denial of Service (DDoS) described in the next paragraph. As shown in Figure 4, we leverage the graph-based data structure to aggregate the network traffic by either the same destination or the source IP. We generate two classes of traffic pattern data, namely, the destination based traffic pattern data that has the same destination IP, and the source based traffic pattern data that has the same source IP. The parameters that we use in the model are given in Table I. The threshold values can be adjusted using a neural network or an optimization algorithm such as Particle Swarm Optimization (PSO) [74].

The proposed Netflow-based anomaly detection approach described in Figure 4 checks whether the flow size of an individual flow is small, the number of packets-per-flow is small, and whether or not a large number of flows appears. In such case, if a small number of source IP traffic is generated and the number of destination ports is high, then that traffic is assumed to be a host scanning. If the fraction of $N(ACK)/N(SYN)$ is small and the traffic pattern data reports a small number of destination ports, this fact implies that the system encounters a TCP SYN flood attack traffic. A further checking is executed, where the source-based traffic pattern data is examined to investigate the traffic sent from a specific host. The detector checks whether the $N(flow)$ is large, the $Avg(flowSize)$ is small, and the $Avg(nPacket)$ is small in a manner similar to destination-based traffic pattern data. The detection approach checks if the data reports a large number of destination IPs and a small number of destination ports. If so, that traffic is suspected to be a flooding attack. Any traffic sent or received from a certain machine is also investigated regardless of the source and destination of traffic pattern data because the system may use network resources by sending or receiving too much traffic when it is used as a flooding attack. Accordingly, the traffic is considered a flooding attack if there is an high bandwidth usage and an high amount of total packets in the traffic data analyzed. Figure 4 shows a flow chart for the detection of the following attacks [75]:

a) DOS and DDoS Attacks Detection: In DDoS attacks, an attacker, obtains information on a target system through scanning. By sending scanning packets to the target, it discovers which systems are working and which services are being offered. There are two types of DDoS, logic and flooding attacks [76]. Our approach only detects the flooding attacks which transmit many spurious packets to the target system, thus wasting CPU, memory, and network resources. In the case of TCP SYN flood [76], the victim receives packets that exceed the buffer of the data structure limit, and cripples its service. Also, the ICMP, TCP, and UDP flooding attacks overwhelm bandwidth by sending useless traffic to the victim.

b) Host Scanning Attacks Detection: During scanning, the attacker makes many connection attempts. Consequently, many flows are generated, and the packet count in each flow is small when a scanning occurs. In addition, the packet size is mostly small (about 40 bytes), because the attacker sends small packets and observes responses from these packets. If an attacker attempts to check open ports in a host, then this host scanning causes traffic with a specific destination IP address.

c) Network Scanning Attacks Detection: A network scanning makes many destination IP addresses when searching for service availability in many hosts of the network. However, the total packet count and total bandwidth can be large or small according to the number of connected hosts and ports. These fields cannot be used to detect scanning.

d) The TCP SYN flood Attack Detection: The TCP SYN flood induces a lot of flow activities, because it sends many packets to a specific port of the victim. Also, the packet count and total packet length in each flow are small, as this attack sends small SYN packets. However, the total bandwidth and total packet count vary according to the number of transmitted packets. Examples for this category of attacks are the Ping-Pong, Smurf, and Fraggle.

e) ICMP, UDP, TCP flooding attacks detection: In addition to the attacks described above, general ICMP, UDP, TCP flooding attacks have dynamic traffic patterns depending on how many packets and hosts are used for an attack. However, most attacks create a large total bandwidth and high total packet count. In addition, such traffic has a small deviation in the packet and flow size of each flow.

However, the proposed Netflow-based anomaly detection approach can only be used to detect particular types of attacks that overload the network traffic. If an attack does not influence the network traffic, it will be difficult to be detected by the proposed approach. Furthermore, this approach uses network driven values for the threshold parameters that are used in the detection process. This makes this approach not applicable to other networks and, therefore, training must be used to set the threshold values based on the parameters of each target network.

Parameter	Description	Thresholds	Description
$N(D\_IP)$	Number of distinguished destination IP with same source IP	$dip-T$	A threshold value represents the maximum normal number of distinguished destination IPs with the same source IP.
$N(S\_IP)$	Number of distinguished source IP with same destination IP	$sip-T$	A threshold value represents the minimum normal number of distinguished source IPs with the same destination IP.
$N(D\_port)$	Number of destination port with same detection IP (destination or source IP)	$dp-LT$, $dp-HT$	Two thresholds represent the minim normal number of destination ports with same detection IP and the maximum one respectively.
$N(flow)$	Number of flows with same detection IP (destination or source IP)	$nf-T$	A threshold value represents the maximum normal number of flows with the same detection IP.
$Sum(flowSize)$, $Avg(flowSize)$	Summation and average of flow size with same detection IP (destination or source IP)	$fs-LT$, $fs-HT$	Two thresholds represent the lowest normal flow size with same detection IP and the highest one respectively.
$Sum(nPacket)$, $Avg(nPacket)$	Summation and average of packet count with same detection IP (destination or source IP)	$np-LT$, $np-HT$	Two thresholds represent the smallest normal number of packets with same detection IP and the highest one respectively.
$N(SYN)$, $N(ACK)$	The total number of SYN and ACK flags with the same destination IP.	$sa-T$	A threshold value represents the minimum normal number of SYN and ACK flags with the same destination IP.

TABLE I
ANOMALY DETECTION PARAMETERS

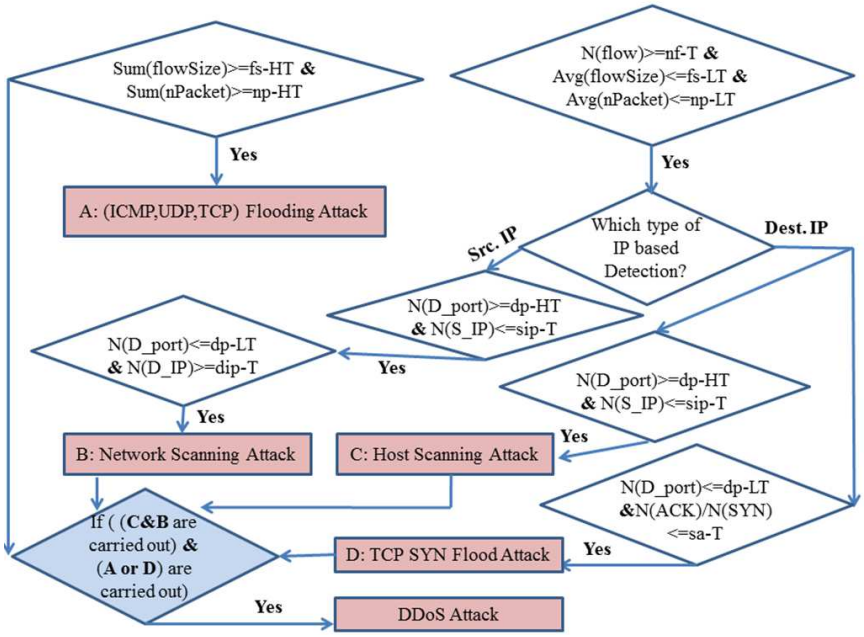

Fig. 4. Flow Chart of the proposed Anomaly Detection Approach

V. CASE STUDY

In the previous sections, we have introduced two methods for the generation of synthetic property-graphs that can capture all the features of a Netflow trace. However, since the main purpose of the synthetic data is to serve as the dataset for a Next-Generation IDS benchmark, we run two sets of experiments to show that: (i) the synthetic dataset is representative of the real data that has been used as the seed, and (ii) the proposed data generation algorithms can produce large amounts of data with reasonable execution time, while also exhibiting a linear strong scalability [77]. Veracity and performance of both PGPBA and PGSK are evaluated.

All the experiments have been executed using a data seed built by the preliminary steps described in Figure 1 on the network trace produced on 10/10/2011 by the Swedish Department of Defense [78]. The size of the seed, measured in terms of number of edges is 1.9 millions (1,940,814). As for the experiment testbed, we used the Shadow II supercomputer [79] hosted by the High Performance Computing Collaboratory (HPC^2) at Mississippi State University (MSU). It is composed of 110 compute nodes, and each one of them has 512GB of RAM, 2 Intel Xeon E5-2680 v2 CPUs with 10

cores each and 54 Gbps InfiniBand network interconnection. Apache Spark 2.1.0 is used as the general big data processing framework for all the experiments.

Since PGPBA and PGSK are probabilistic algorithms, the number of vertices and edges generated at each run of the experiment can slightly vary. Furthermore, PGSK's edge generation rate is exponential in the number of iterations, while PGPBA's edge generation rate is linear in the number of iterations and proportional to the specified fraction parameter. As a consequence, since we don't have a fine grain control on the size of the produced graphs, instead of focusing on a point-based comparison of veracity and performance, we will evaluate their trends while trying at the same time to generate graphs of similar sizes.

A. Veracity Evaluation

The evaluation of the veracity of the synthetic datasets is carried out by analyzing their degree and pagerank distributions. Specifically, in order to compare the seed and the synthetic graphs in the same ranges of values, we use the normalized degree and pagerank distributions. The latter are computed by dividing each degree and pagerank value, respectively, by the sum of the degrees and by the sum of the pagerank values of all the vertices.

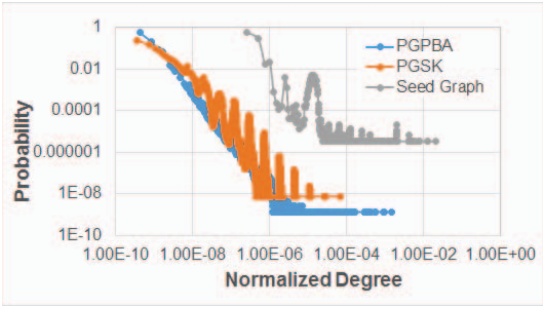

Fig. 5. Comparison of the degree distributions

Figure 5 compares the degree distribution exhibited by the seed graph, which counts almost 2 millions of edges, with the ones exhibited by the synthetic graphs generated with PGPBA and PGSK, counting, respectively, 1.15 and 1.34 billions of edges. All the three distributions have similar shapes, but PGSK exhibits more spikes. This is due to the fact that PGSK is based on Kronecker, and therefore it tends to replicate many times the same exact graph structure, which contains a single evident spike in the seed graph. It is also worth noting that the seed distribution is not overlapping with the synthetic graphs' distribution. This is due to the normalization: the synthetic graphs are roughly 3 orders of magnitude larger than the seed, therefore causing a shift to down-left of three orders of magnitude in the degree distributions.

While the plot in Figure 5 can give a good indication about the veracity of the generated data, we also want to study how the veracity varies according to the size of the generated graphs. To this end, we define the *veracity score* of a synthetic dataset with respect to the seed dataset as the average Euclidean distance of their normalized degree and PageRank distributions. A smaller veracity score indicates higher similarity with the seed dataset. In the following experiments, we compare the veracity scores of PGPBA and PGSK according to the size, measured in the number of edges of the generated data.

Figure 6 shows the veracity scores of the degree distributions. The degree and PageRank veracity scores obtained by PGSK were in the range $[5.26 \times 10^{-10}, 6.37 \times 10^{-3}]$ and $[5.08 \times 10^{-25}, 4.25 \times 10^{-18}]$, respectively. The degree and PageRank veracity score ranges of PGPBA, with fraction parameters 0.1, 0.3, 0.6 and 0.9, are listed below:

1) **Fraction 0.1**: $[5.24 \times 10^{-10}, 2.21 \times 10^{-7}]$, $[5.70 \times 10^{-25}, 1.04 \times 10^{-22}]$
2) **Fraction 0.3**: $[2.72 \times 10^{-10}, 1.73 \times 10^{-7}]$,$[1.34 \times 10^{-25}, 2.58 \times 10^{-23}]$
3) **Fraction 0.6**: $[5.02 \times 10^{-10}, 3.25 \times 10^{-7}]$,$[1.35 \times 10^{-25}, 6.20 \times 10^{-23}]$
4) **Fraction 0.9**: $[5.57 \times 10^{-10}, 2.49 \times 10^{-7}]$,$[1.34 \times 10^{-25}, 1.37 \times 10^{-23}]$

All five experiments exhibit a linear veracity score (vertical axis) which decreases when the size of the graph (i.e., number of edges in the horizontal axis) grows. Note that the minimum veracity scores are obtained at the maximum number of edges. When the synthetic graph is relatively small, it does not hold enough information to reflect the original data distribution in the seed graph. However, as the size of the synthetic graph keeps growing, there is increasingly more capability to inherit the properties of the seed graph. The same trend also applies to the PageRank distribution in Figure 7. It is worth noting that, for both the veracity scores in Figure 6 and 7, the PGPBA curve starts with a number of edges equal to approximately 5 millions, while PGSK starts with as low as 100 edges. This is due to the nature of PGSK. By using a 2×2 initiator matrix computed with the Kronfit procedure, the PGSK can generate graphs which are smaller than the seed graph.

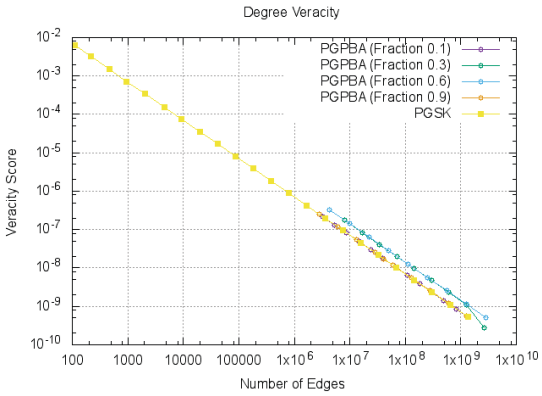

Fig. 6. Evaluation of Degree Veracity

In general, PGPBA and PGSK exhibit comparable degree veracities, when the former is configured with a fraction

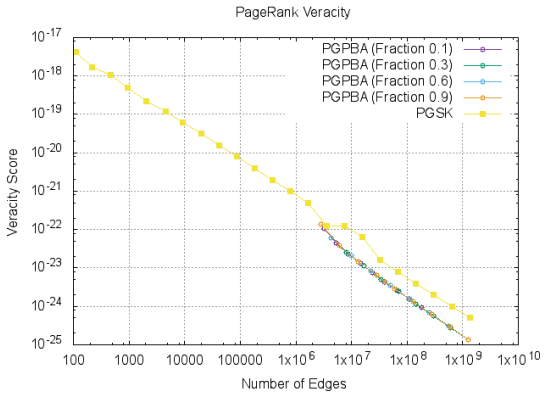

Fig. 7. Evaluation of PageRank Veracity

parameter equal to 0.1. Bigger fraction values allow PGPBA to generate larger graphs with less iterations, but with some loss of precision in rendering the degree distribution. Regarding the PageRank degree distributions, PGPBA clearly performs better in all the cases. However, we also note that the veracity score obtained in both the experiments are in general very low, meaning that the synthetic dataset properly mimics the characteristics of the seed in all the configurations of the PGPBA and PGSK.

B. Performance

In this section, we empirically demonstrate the linear scalability of the proposed Map-Reduce implementations of PGPBA and PGSK algorithms. For the purpose of our experiments, we used up to 55% of Shadow II, that is, we ran experiments with a number of compute nodes ranging between 10 and 60. Specifically, we carried out two sets of experiments, aimed respectively at showing the scalability of the algorithms and the scalability of the platform. For the former, we generate graphs of increasing size with a fixed amount of computational resources. For the latter, we generate graphs of fixed size with a varying amount of computational resources. During all the experiments we also collected the memory utilization of all the compute nodes.

There are many configuration parameters and tuning possibilities for Spark. In order to make the experiments easily reproducible, we only changed a few of them, namely:

- the number of executor cores `total-executor-cores`, which specifies the maximum number of physical cores usable by the platform, and corresponds to the maximum number of spawned threads. The threads are equally allocated among the available compute nodes if Spark is configured as a standalone cluster, as in our case;
- the amount of memory available to the worker nodes `--executor-memory`;
- the amount of memory available to the cluster driver `--driver-memory`;
- the number of partitions in which the data is divided. This is an application-level parameter, that we introduced to have a finer control on the performance. We found that, in most cases, using a number of partitions equal to $2\times$ or $4\times$ the number of executor cores leads to the best performance.

The `total-executor-cores` parameter has been object of a preliminary study aimed at finding its optimal setting.

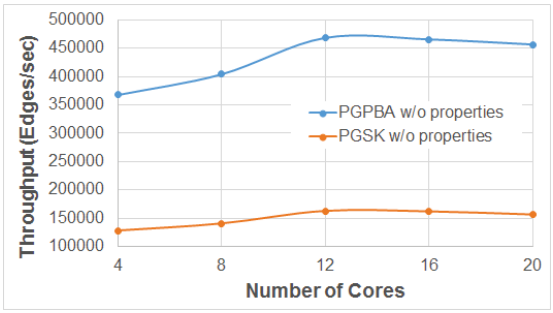

Fig. 8. Single Node Throughput

Figure 8 shows that the maximum throughput for both PGPBA and PGSK can be achieved with 12 active cores out of the available 20. That is, there is no performance increase in using the remaining cores.

For this reason, in the following experiments we always set `total-executor-cores` to $12\times$ the number of compute nodes, and the number of partitions to $2\times$ the number of executor cores. The memory parameters are fixed to $450GB$ for both the driver and the workers, so that a small portion of memory can be left for the operating system.

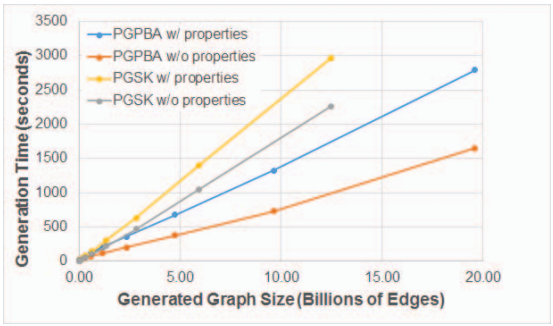

Fig. 9. Edges Generation Time Comparison of PGPBA and PGSK

Figure 9 shows the comparison of the generation time of PGPBA and PGSK. Since the Kronecker algorithm doubles the size of the graph at each iteration, for a fair comparison between PGPBA and PGSK, we set PGPBA's $fraction = 2$. We used 60 Shadow compute nodes and we generated graphs of sizes ranging between 4 millions and 20 billions of edges. It is clear that the generation time of both algorithms is linear, according to the size of the generated graph. However, PGPBA provides better performances, as also highlighted in the throughput plot shown in Figure 10. The generation of

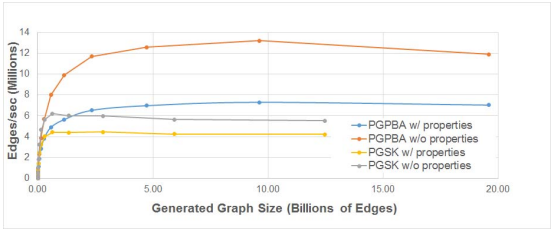

Fig. 10. Edges Generation Throughput Comparison of PGPBA and PGSK

the properties for vertices and edges introduces an average overhead of 50% in the case of PGPBA, and of 30% for PGSK. However, it is worth noting that the function for the generation of the properties is the same in both synthesis methods, thus the greater impact on PGPBA is only due to its lower generation time.

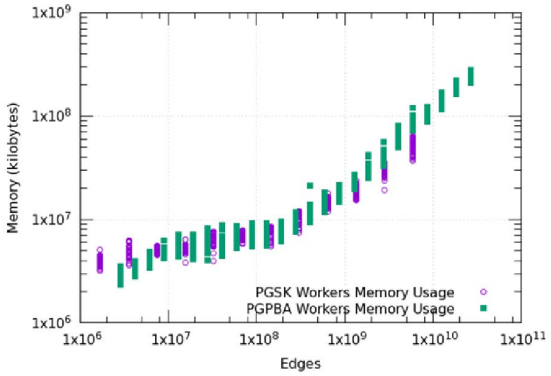

Fig. 11. Memory Usage Comparison of PGPBA and PGSK

Figure 11 shows the comparison of the memory usage of the worker nodes when executing PGPBA and PGSK to generate graphs of varying size. We can divide the plot in two ranges: $[1 \times 10^6, 1 \times 10^8]$ and $[1 \times 10^8, 1 \times 10^{11}]$. In the former, the memory usage of the worker nodes is almost constant, barely reaching $10GB/compute\ node$. This is due to the overhead introduced by the platform, which is not negligible for the generation of small graphs. In the right side of the plot, instead, it is possible to note a linear increase in the memory usage, reaching up to almost $300GB/compute\ node$ for graphs with 20 billions of edges.

The strong scalability study of the platform is reported in Figure 12, where we measure the speedup obtained by generating graphs of fixed size with an increasing number of compute nodes, ranging from 10 to 60. We generate graphs with the largest size that is possible to handle with 10 compute nodes of Shadow II, which is equal to 9.6 billions of edges for PGPBA, and 6 billions of edges for PGSK, corresponding to 12 iterations with $fraction = 2$ for the former, and 30 iterations for the latter. PGPBA exhibits a linear strong scalability and it is very close to the ideal speedup. PGSK also exhibits a linear scalability, but it is more distant from the ideal speedup with respect to PGPBA, therefore confirming that, in general, the latter provides the best performance.

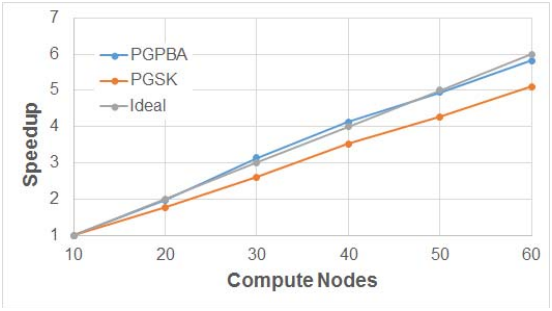

Fig. 12. Speedup comparison of PGPBA and PGSK

VI. Conclusions and Future Works

Benchmarking graph-based databases and distributed graph processing platforms is important to understand the performance of next-generation graph-based applications. IDSs are gradually adopting graph-based data structures and, so far, no graph-based benchmarks have been made available to measure their performance. A domain-specific dataset is one of the main components of a benchmark. However, very few publicly available datasets exist for cyber-security applications due to privacy concerns.

In this paper, we address the problem of generating datasets based on property-graphs and representing network traces. To this end, we extend two well-known models for synthetic graph generation, namely, Barabási-Albert and Kronecker. The resulting novel approaches, named respectively Property-Graph Parallel BA (PGPBA) and Property-Graph Stochastic Kronecker (PGSK), guarantee high veracity or, in other words, a high similarity degree between some network trace used as seed and the generated synthetic data. Furthermore, we experimentally show that PGPBA and PGSK scale linearly according to the size of the synthetic graph and to the number of compute nodes available for the execution of the algorithms. The data generators presented are part of a benchmarking suite, which we released with the GPLv3 open-source license and that is available at the URL: https://github.com/msstate-dasi/csb.

The proposed approach can help researchers and system administrators in modeling and evaluating the performance of next-generation IDSs, therefore possibly allowing them to precisely quantify the time-to-detection of network threats. However, it cannot be currently used as a fully functional IDS because it only implements the algorithms that can be used to perform intrusion detection, rather than a full intrusion detection process. As future work, we plan to extend the platform to fully support off-line intrusion detection, followed by on-line intrusion detection with streaming data.

Acknowledgment

Funding for this work was (partially) provided by the Pacific Northwest National Laboratory, under U.S. Department of Energy Contract DE-AC05-76RL01830.

REFERENCES

[1] "The bro network security monitor," 2016. [Online]. Available: https://www.bro.org

[2] "Snort - network intrusion detection," 2017. [Online]. Available: https://www.snort.org

[3] "Alert correlation, assessment and reaction module - next generation," 2009. [Online]. Available: http://www.acarm.wcss.wroc.pl

[4] "Aide - advanced intrusion detection environment," 2016. [Online]. Available: http://aide.sourceforge.net

[5] W. Wang, *A graph oriented approach for network forensic analysis*. Iowa State University, 2010.

[6] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *Computers & Security*, vol. 31, no. 3, 2012.

[7] E. Casey, "Investigating sophisticated security breaches," *Communications of the ACM*, vol. 49, no. 2, pp. 48–55, 2006.

[8] B. Foo, Y.-S. Wu, Y.-C. Mao, S. Bagchi, and E. Spafford, "Adepts: adaptive intrusion response using attack graphs in an e-commerce environment," in *2005 International Conference on Dependable Systems and Networks (DSN'05)*. IEEE, 2005, pp. 508–517.

[9] N. Stakhanova, S. Basu, and J. Wong, "A cost-sensitive model for preemptive intrusion response systems." in *AINA*, vol. 7, 2007.

[10] T. Toth and C. Kruegel, "Evaluating the impact of automated intrusion response mechanisms," in *Computer Security Applications Conference, 2002. Proceedings. 18th Annual*. IEEE, 2002, pp. 301–310.

[11] "Open source implementation of mapreduce," 2017. [Online]. Available: http://hadoop.apache.org

[12] J. G. Shanahan and L. Dai, "Large scale distributed data science using apache spark," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015, pp. 2323–2324.

[13] Y. Huai, A. Chauhan, A. Gates, G. Hagleitner, E. N. Hanson, O. O'Malley, J. Pandey, Y. Yuan, R. Lee, and X. Zhang, "Major technical advancements in apache hive," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 2014, pp. 1235–1246.

[14] R. S. Xin, J. Rosen, M. Zaharia, M. J. Franklin, S. Shenker, and I. Stoica, "Shark: Sql and rich analytics at scale," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of data*. ACM, 2013, pp. 13–24.

[15] R. Han, Z. Jia, W. Gao, X. Tian, and L. Wang, "Benchmarking big data systems: State-of-the-art and future directions," *CoRR*, vol. abs/1506.01494, 2015.

[16] A. Ghazal, T. Rabl, M. Hu, F. Raab, M. Poess, A. Crolotte, and H.-A. Jacobsen, "Bigbench: towards an industry standard benchmark for big data analytics," in *Proceedings of the 2013 ACM SIGMOD international conference on Management of data*. ACM, 2013, pp. 1197–1208.

[17] L. Wang, J. Zhan, C. Luo, Y. Zhu, Q. Yang, Y. He, W. Gao, Z. Jia, Y. Shi, S. Zhang *et al.*, "Bigdatabench: A big data benchmark suite from internet services," in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2014.

[18] F. Cohen, "Simulating cyber attacks, defences, and consequences," *Computers & Security*, vol. 18, no. 6, pp. 479–518, 1999.

[19] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *science*, vol. 286, no. 5439, pp. 509–512, 1999.

[20] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani, "Kronecker graphs: An approach to modeling networks," *Journal of Machine Learning Research*, vol. 11, no. Feb, pp. 985–1042, 2010.

[21] T. Frisendal, *Graph Data Modeling for NoSQL and SQL: Visualize Structure and Meaning*. Technics Publications, LLC, 2016.

[22] B. Claise, "Cisco systems netflow services export version 9," 2004. [Online]. Available: https://tools.ietf.org/html/rfc3954

[23] "Libpcap," 2017. [Online]. Available: https://sourceforge.net/projects/libpcap/

[24] J. Webber, "A programmatic introduction to neo4j," in *Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity*. ACM, 2012, pp. 217–218.

[25] "Titan: distributed graph database," 2017. [Online]. Available: https://github.com/thinkaurelius/titan

[26] N. Martínez-Bazan, V. Muntés-Mulero, S. Gómez-Villamor, J. Nin, M.-A. Sánchez-Martínez, and J.-L. Larriba-Pey, "Dex: high-performance exploration on large graphs for information retrieval," in *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*. ACM, 2007, pp. 573–582.

[27] J. Webber, "Graph processing versus graph databases," 2017. [Online]. Available: http://jimwebber.org/2011/08/graph-processing-versus-graph-databases/

[28] R. S. Xin, J. E. Gonzalez, M. J. Franklin, and I. Stoica, "Graphx: A resilient distributed graph system on spark," in *First International Workshop on Graph Data Management Experiences and Systems*. ACM, 2013, p. 2.

[29] S.-H. Lim, S. Lee, G. Ganesh, T. C. Brown, and S. R. Sukumar, "Graph processing platforms at scale: Practices and experiences," in *Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium on*. IEEE, 2015, pp. 42–51.

[30] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Tech. Rep., 1999.

[31] J. Leskovec and E. Horvitz, "Planetary-scale views on a large instant-messaging network," in *Proceedings of the 17th international conference on World Wide Web*. ACM, 2008, pp. 915–924.

[32] J. Yang and J. Leskovec, "Defining and evaluating network communities based on ground-truth," *Knowledge and Information Systems*, vol. 42, no. 1, pp. 181–213, 2015.

[33] M. Girvan and M. E. Newman, "Community structure in social and biological networks," *Proceedings of the national academy of sciences*, vol. 99, no. 12, pp. 7821–7826, 2002.

[34] G. Siganos, M. Faloutsos, P. Faloutsos, and C. Faloutsos, "Power laws and the as-level internet topology," *IEEE/ACM Transactions on Networking (TON)*, vol. 11, no. 4, pp. 514–524, 2003.

[35] A.-L. Barabási, "Scale-free networks: a decade and beyond," *science*, vol. 325, no. 5939, pp. 412–413, 2009.

[36] P. Erdos and A. Rényi, "On the evolution of random graphs," *Publ. Math. Inst. Hung. Acad. Sci*, vol. 5, no. 1, pp. 17–60, 1960.

[37] B. Bollobás, "Random graphs," in *Modern Graph Theory*. Springer, 1998, pp. 215–252.

[38] P. W. Holland, K. B. Laskey, and S. Leinhardt, "Stochastic blockmodels: First steps," *Social networks*, vol. 5, no. 2, pp. 109–137, 1983.

[39] O. Frank and D. Strauss, "Markov graphs," *Journal of the american Statistical association*, vol. 81, no. 395, pp. 832–842, 1986.

[40] G. Robins, P. Pattison, Y. Kalish, and D. Lusher, "An introduction to exponential random graph (p*) models for social networks," *Social networks*, vol. 29, no. 2, pp. 173–191, 2007.

[41] D. Chakrabarti, Y. Zhan, and C. Faloutsos, "R-mat: A recursive model for graph mining," in *Proceedings of the 2004 SIAM International Conference on Data Mining*. SIAM, 2004, pp. 442–446.

[42] J. Leskovec and C. Faloutsos, "Scalable modeling of real graphs using kronecker multiplication," in *Proceedings of the 24th international conference on Machine learning*. ACM, 2007, pp. 497–504.

[43] D. J. Watts and S. H. Strogatz, "Collective dynamics of small-worldnetworks," *nature*, vol. 393, no. 6684, pp. 440–442, 1998.

[44] F. Chung and L. Lu, "Connected components in random graphs with given expected degree sequences," *Annals of combinatorics*, vol. 6, no. 2, pp. 125–145, 2002.

[45] F. Chung and L. Lu, "The average distance in a random graph with given expected degrees," *Internet Mathematics*, vol. 1, no. 1, 2004.

[46] C. Seshadhri, T. G. Kolda, and A. Pinar, "Community structure and scale-free collections of erdős-rényi graphs," *Physical Review E*, vol. 85, no. 5, p. 056109, 2012.

[47] T. G. Kolda, A. Pinar, T. Plantenga, and C. Seshadhri, "A scalable generative graph model with community structure," *SIAM Journal on Scientific Computing*, vol. 36, no. 5, pp. C424–C452, 2014.

[48] J. Leskovec, *Dynamics of large networks*. Carnegie Mellon University, 2008.

[49] M. Alam, M. Khan, A. Vullikanti, and M. Marathe, "An efficient and scalable algorithmic method for generating large–scale random graphs," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Press, 2016, p. 32.

[50] A. Yoo and K. Henderson, "Parallel generation of massive scale-free graphs," *arXiv preprint arXiv:1003.3684*, 2010.

[51] "Graph 500," 2016. [Online]. Available: http://graph500.org

[52] L. Wang, C. Yao, A. Singhal, and S. Jajodia, "Implementing interactive analysis of attack graphs using relational databases," *Journal of Computer Security*, vol. 16, no. 4, pp. 419–437, 2008.

[53] S. Jajodia, S. Noel, P. Kalapa, M. Albanese, and J. Williams, "Cauldron mission-centric cyber situational awareness with defense in depth," in

Military Communications Conference, 2011-MILCOM 2011. IEEE, 2011, pp. 1339–1344.
[54] E. Bou-Harb and M. Scanlon, "Behavioral service graphs: A formal data-driven approach for prompt investigation of enterprise and internet-wide infections," *Digital Investigation*, vol. 20, pp. S47–S55, 2017.
[55] C. C. Noble and D. J. Cook, "Graph-based anomaly detection," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2003, pp. 631–636.
[56] P.-Y. Chen and A. O. Hero, "Assessing and safeguarding network resilience to nodal attacks," *IEEE Communications Magazine*, vol. 52, no. 11, pp. 138–143, 2014.
[57] L. Akoglu, H. Tong, and D. Koutra, "Graph based anomaly detection and description: a survey," *Data Mining and Knowledge Discovery*, vol. 29, no. 3, pp. 626–688, 2015.
[58] J. McGlone, A. Marshall, and R. Woods, "An attack-resilent sampling mechanism for integrated ip flow monitors," in *Distributed Computing Systems Workshops, 2009. ICDCS Workshops' 09. 29th IEEE International Conference on*. IEEE, 2009, pp. 233–238.
[59] W. Yang, J. Gong, W. Ding, and X. Wu, "Network traffic emulation for ids evaluation," in *Network and Parallel Computing Workshops, 2007. NPC Workshops. IFIP International Conference on*. IEEE, 2007.
[60] Cisco, "Cisco router firewall security: Dos protection," 2004. [Online]. Available: https://goo.gl/Aqf8AM
[61] M. Fullmer and S. Romig, "The osu flowtools package and cisco netflow logs," in *Proceedings of the 2000 USENIX LISA Conference*, 2000.
[62] SANS, "Intrusion detection through relationship analysis," 2016. [Online]. Available: https://goo.gl/WNKrcH
[63] Oracle, "Using property graphs in a big data environment," 2017. [Online]. Available: https://goo.gl/G6F3Qe
[64] F. C. Smith and E. E. Kenneally, "Electronic evidence and digital forensics testimony in court," in *Handbook of Digital and Multimedia Forensic Evidence*. Springer, 2008, pp. 103–132.
[65] R. Albert and A.-L. Barabási, "Statistical mechanics of complex networks," *Reviews of modern physics*, vol. 74, no. 1, p. 47, 2002.
[66] M. van Steen, *Graph Theory and Complex Networks: An Introduction*. On Demand Publishing, LLC-Create Space, 2010.
[67] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," *Computer networks and ISDN systems*, vol. 30, no. 1, pp. 107–117, 1998.
[68] A. M. Z. Bidoki and N. Yazdani, "Distancerank: An intelligent ranking algorithm for web pages," *Information Processing & Management*, vol. 44, no. 2, pp. 877–892, 2008.
[69] M. Barthelemy, "Betweenness centrality in large complex networks," *The European Physical Journal B-Condensed Matter and Complex Systems*, vol. 38, no. 2, pp. 163–168, 2004.
[70] D. S. Hirschberg, A. K. Chandra, and D. V. Sarwate, "Computing connected components on parallel computers," *Communications of the ACM*, vol. 22, no. 8, pp. 461–464, 1979.
[71] V. Paxson, "Bro: a system for detecting network intruders in real-time," *Computer networks*, vol. 31, no. 23, pp. 2435–2463, 1999.
[72] A. A. Galtsev and A. M. Sukhov, "Network attack detection at flow level," in *Smart Spaces and Next Generation Wired/Wireless Networking*. Springer, 2011, pp. 326–334.
[73] M.-S. Kim, H.-J. Kong, S.-C. Hong, S.-H. Chung, and J. W. Hong, "A flow-based method for abnormal network traffic detection," in *Network operations and management symposium, 2004. NOMS 2004. IEEE/IFIP*, vol. 1. IEEE, 2004, pp. 599–612.
[74] H. Wang, H. Sun, C. Li, S. Rahnamayan, and J.-S. Pan, "Diversity enhanced particle swarm optimization with neighborhood search," *Information Sciences*, vol. 223, pp. 119–135, 2013.
[75] H. A. Kholidy, F. Baiardi, S. Hariri, E. Elhariri, A. Yousof, and S. Shehata, "A hierarchical cloud intrusion detection system: Design and evaluation," *International Journal on Cloud Computing: Services and Architecture (IJCCSA)*, vol. 2, no. 6, 2012.
[76] D. Moore, C. Shannon, D. J. Brown, G. M. Voelker, and S. Savage, "Inferring internet denial-of-service activity," *ACM Transactions on Computer Systems (TOCS)*, vol. 24, no. 2, pp. 115–139, 2006.
[77] D. M. Pressel, "Scalability vs. performance," DTIC Document, Tech. Rep., 2001.
[78] "Swedish department of defense dataset," 2011. [Online]. Available: ftp://download.iwlab.foi.se/dataset/smia2011/Network_traffic/
[79] MSU, "Hpc: High performance computing laboratory," 2016. [Online]. Available: https://www.hpc.msstate.edu/computing/hpc.php