

Generation of SSH Network Traffic Data for IDS Testbeds

Hristo Djidjev *

Los Alamos National Laboratory, Los Alamos, New Mexico, U.S.A.

Lyudmil Aleksandrov

Institute of Information and Communication Technologies, Sofia, Bulgaria

Abstract

We develop an algorithm for generating secure shell (ssh) network traffic that can find use as a part of a testbed for evaluating anomaly detection and intrusion detection systems in cyber security. Given an initial dataset describing real network traffic, the generator produces synthetic traffic with characteristics close to the original. The objective is to match parameters of the original traffic such as traffic volumes, session durations, diurnal patterns, and relationships between hosts in terms of communicating pairs and subsets.

1 Introduction

One of the most challenging tasks of cyber security is preventing the unauthorized access to computer systems and the stored information of commercial or national security value. Intrusion detection systems (IDSs) monitor the traffic and try to discover evidence of malicious activities. One approach for achieving that is to use the methods and tools of anomaly detection, which try to discover patterns of activity that are unusual (anomalous). Such IDSs work in two modes. In the *training mode*, the historical traffic is analyzed and a model of normal behavior is created. Such model might contain typical values or statistical distributions of some statistical parameters of the traffic and the threshold beyond which the traffic will be labeled as anomalous. In the *detection phase*, the new (current) traffic is analyzed and compared to the model. If the traffic is determined to be anomalous, it is then forwarded to a cyber security specialist for additional analysis.

In order to test existing and new anomaly and intrusion detection algorithms, it is often preferable to use a generator that can produce network traffic that resembles

a real observed one and that uses adjustable parameters such as the number of hosts, number of sessions, and average session durations. Using real traffic for testing purposes has several drawbacks. Network data can be difficult or impossible to obtain for many researchers. Sharing data between researchers can pose security or privacy issues or be forbidden by organization's cyber policies. For anomaly detection testing purposes, it will be hard to distinguish between normal and malicious real traffic, and therefore, it will be hard to evaluate the performance of the tested anomaly detection tool. Using a traffic generator allows one to mix normal and malicious traffic in various scenarios and also vary the other traffic parameters so that the anomaly detector can be tested under different circumstances.

In this paper we describe a new algorithm for generating normal (as opposed to malicious) network traffic based on the secure shell (ssh) protocol. The traffic data we generate is in the form of a set of ssh sessions, where for each session we provide its source and target nodes (corresponding to the IP addresses of hosts in the real traffic), its start time, and its duration. Our goal is to capture patterns of real traffic associated with relationships between pairs of hosts and pairs of sessions. Specifically, we are interested in interrelated sessions that may have been run by the same user during certain activity. For instance, a legitimate user may start at host *A*, ssh to host *B*, ssh from host *B* to host *C* (as host *C* may not be directly reachable from *A*), and then close all sessions. In such a case we consider sessions *AB* and *BC* related. We merge such interrelated sessions in structures we call telescoping subgraphs (TSGs) [3]. More formally, TSGs are maximal subgraphs of the traffic graph that satisfy the telescoping property, illustrated on Figure 1(a). (Maximal means here that the subgraph cannot be expanded with more edges satisfying the telescoping property.) A decomposition of a traffic graph into TSGs, called TSG-decomposition, can be computed in nearly

*The authors acknowledge and appreciate the support provided for this work by the Los Alamos National Laboratory Directed Research and Development Program (LDRD).

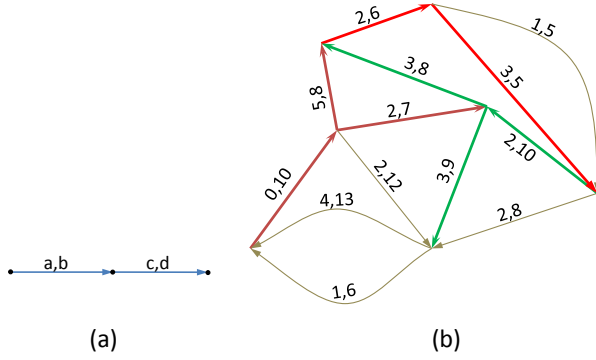


Figure 1: A telescoping subgraph (TSG). Edges have labels s, e , where s is the start time and e is the end time of the corresponding sessions. (a) Two edges with labels a, b and c, d can appear consecutively in a TSG only if they satisfy $a \leq c$ and $b \geq d$ (the *telescoping property*). (b) TSG-decomposition. The traffic graph can be decomposed into maximal TSGs so that each edge belongs to some TSG. TSGs with more than one edge are shown with different colors. All remaining TSGs are single-edge ones and are shown in the same color.

linear time [3].

The difference between the patterns of legitimate users and intruders has proved useful and has been exploited in previous anomaly detectors [3, 6]. Hence, one of our objectives is to generate traffic whose hosts and sessions are interrelated in a way resembling the one observed in the original traffic. Other objectives are to preserve the temporal aspects of the traffic, namely, the varying levels of activity during different days of the week and times of the day, the session durations, the total volume of the traffic, and the distributions of the node *outdegrees* and *indegrees*, defined as the numbers of sessions with origin or target a given host, respectively. Finally, we want our algorithm to be scalable so that it will be able to generate traffic with tens of thousands of hosts and millions of sessions in a reasonable time, e.g., in an hour or less.

2 Related work

One of the most comprehensive efforts to generate realistic network traffic was by the DARPA sponsored project on developing an evaluation system for IDS performed by the MIT Lincoln Laboratory [4]. Their system is agent-based, where software programs (agents) simulate the activities of real users. While that method can recreate realistic traffic volumes and application usage patterns, it cannot emulate the topology of the underlying network and the communication patterns between hosts and servers.

Several more recent works that take into account the network topology have targeted generation of web usage (Internet) traffic. Sommers et al. [7] describe a network flow generation tool called *fs* that generates traffic for arbitrary network topology. Vishwanath and Vahdat [10] describe a traffic generator called *Swing* that captures packet interactions between applications and users as well as burstiness of traffic. However, in these and other similar projects, e.g. *tmix* [11], the network topology is specified by the user and is not re-generated based on an analysis of historical data. Tools such as *Monkey* [1] that do use empirical trace data, emulate only behavior at local host level and fail to analyze and emulate the global communications data patterns.

We are not aware of any other generation tool for cybersecurity applications such as ours that can recreate the global network topology and the global communication patterns, while not exactly and explicitly copying the real network. A similar network generation problem has been addressed from purely graph-theoretic viewpoint, i.e., generating a random network (graph) that has expected degrees matching ones of a given graph [2]. However, that method does not take into account the temporal information and relationships between edges (sessions) that is key to our study. Finally, a problem that is not a subject of this paper, generating malicious traffic, has been discussed in several works, including [5, 8].

3 Algorithm outline

The structure of our generation algorithm can be seen as being analogous to the anomaly detection algorithm from [3]. Both algorithms have a first phase that processes historical data and extract a model for the traffic, although the information in that model is a bit different in the two cases. The information that is used in the generation algorithm, called *profile* of the traffic, includes the indegrees and outdegrees of the nodes, the number and sizes of the TSGs, the dependencies between edges, i.e., which pairs of edges occur consecutively in the same TSG, and information about session durations. That information is collected over a certain period of time (time window) immediately preceding the current time in a *profiling* phase, which corresponds to the training phase in anomaly detection. The time window length W is a run-time parameter currently set at 28 days.

Moreover, both the anomaly detection and the generation algorithm have a second phase that uses the model created in the first one. For anomaly detection, it is the detection phase, where new traffic is compared and evaluated with respect to the model. In the generation algorithm, we have a *generation* phase, where new traffic is generated based on the model (profile). After new traffic is generated covering certain period of time w (in our

implementation w is one hour), the new traffic is added to the model and the traffic from the oldest hour is removed, thereby sliding the time window forward by one hour. Similarly, traffic is generated for another hour, the statistics are updated, and so on until the required duration for the generated traffic is met.

Since the procedure outlined above requires existing traffic covering a period of at least W prior to the current time, the question arises about how to provide an initial traffic of length W . One way will be to just use real traffic of such length in order to generate the new traffic, and at the end of the generation to exclude it from the final result. The disadvantage of this approach is that one will not be able to freely choose some parameters of the synthetic traffic such as number of nodes or number of edges (volume of traffic) as they will be determined by the real traffic that we have used. Furthermore, there is the risk of overfitting the model of the new traffic on the model of the old one. Hence, our generation phase contains two modes; in the *seeding* mode, it generates traffic for duration W based on the profile of the original traffic; in the second, *generation* mode, it generates synthetic traffic based on the current profile and then, each time than new traffic of duration w has been created, it updates the profile taking into account that newest traffic.

To summarize, the structure of our traffic generating algorithm is the following. During the profiling phase, the original traffic is analyzed and an initial profile P is computed. During the generation phase, in a seeding mode, an initial traffic covering a time period of length W is generated based on P , and P is updated by using that new traffic. Then in generation mode, a new traffic of length $w \ll W$ is generated based on (the updated) P , and P is updated including the traffic from the newest window and dropping the oldest window- w traffic. (Hence, the profile is always based on a time window of length W .) The process is repeated, each time adding new traffic of time-length w , until synthetic traffic of desired total time is produced. We will describe each of the phases in more details in the next sections.

4 The profiling phase

The purpose of this phase is to construct a profile of the original traffic that will be used during the generation phase. That profile should contain all the information needed by the generation phase and it has to be organized in a data structure that will allow easy updates when newly generated traffic is taken into account.

Formally, we define a *traffic profile* as a tuple $P = \{num, sz, ptn, dur, n, w, W\}$, where

- num is the total number of TSGs;
- $sz(k)$ is the number of TSGs of size k ;

- ptn gives information about connection patterns for sets of nodes as follows: $ptn(1, e)$ is the number of times edge e has been observed (i.e., how many times the endpoints of e have been connected) and $ptn(2, e, h)$ is the number of times edge h has been preceded by edge e in a TSG;
- $dur(e)$ is the duration of edge e ;
- n is the number of the nodes;
- w, W are the small and the big time windows lengths.

Furthermore, num , sz , and ptn are additionally indexed (stored as separate values) for each of the seven weekdays and for each of the 24 hour combinations, thereby allowing us to capture daily and weekly time patterns during the generation phase. For instance, $num[w, h]$ will denote the total number of TSGs for weekday w and time of the day h , while $ptn(w, h, 1, e)$ is the number edge e has been observed on weekday w at time h .

Our choice of the patterns ptn given above is dictated by our objective to capture the communication relationships between nodes; namely $ptn(1, e)$ contains information about which pairs of nodes have communicated with each other, while $ptn(2, e, h)$ can be used to determine how likely it is that a session by the nodes represented by e is followed by a session corresponding to the endpoints of h . One could possibly consider patterns of more than two edges, but such patterns will increase significantly the size of the profile and the computational complexity of the generation algorithm and decrease its efficiency.

The procedure is outlined in Algorithm 1. First we adjust the number of nodes of G to match the target node number N . If $N > n$, in line 5 we create a new node, add it to the graph, connect it to the rest of the graph by new edges, and repeat until the graph has N nodes in total. In order to define the new edges connecting those new nodes to the rest of G , for each new node v , we choose randomly and uniformly a node w already in G (including both original and newly added nodes), and for each directed edge (w, x) or (y, w) in G define a new directed edge (v, x) or (y, v) , respectively. Finally, start time and duration times of the new edges are assigned randomly by sampling normal distributions with means the corresponding times of the corresponding original edges.

Next, in the loop in lines 7–15, for each time window w , we construct the graph corresponding to traffic that started in that window, and then extract length-1 and length-2 patterns and TSGs from the resulting graph. In order to find those patterns efficiently, we define an auxiliary graph H . H has a vertex v for each edge $e = e(v)$ of G and two vertices u and w of H are connected by

Algorithm 1 Computing initial profile P

Input: Time windows $W > 0$ and $w > 0$, traffic graph G of n nodes and duration $U = kw$, number N of nodes for generated graph

Output: A traffic profile $P = \{num, sz, ptn, dur, N, w, W\}$ for G

```
1: if  $N < n$  then
2:   choose  $n - N$  nodes from  $G$  and remove them
3: end if
4: if  $N > n$  then
5:   create  $N - n$  new nodes and add them to  $G$ 
6: end if
7: for all  $t_0 \in \{0, w, 2w, \dots, (k-1)w\}$  do
8:   for all  $t_1 \in [0, 1, \dots, w]$  do
9:      $t = t_0 + t_1$ 
10:    Define  $\hat{G}$  as the graph containing all edges
        of  $G$  with start time in  $[t, t + w - 1]$ .
11:    Update the number of length-1 and
        lengths-2 patterns in the  $W$ -window by
        removing patterns containing expired
        edges, i.e., edges whose end time is less
        than  $t - W$ .
12:    Find all length-1 and lengths-2 patterns in
         $\hat{G}$  and update the corresponding  $ptn$ 
        and  $dur$  data structures
13:    Find all TSGs of  $\hat{G}$  and use their sizes to
        update the  $num$  and  $sz$  structures
14:   end for  $t_1$ 
15: end for  $t_0$ 
16: Output  $P$ 
```

an edge of H if the target of $e(u)$ is source of $e(w)$ and $e(u)$ and $e(w)$ satisfy the telescoping property. Hence, the edges of H can easily be created by looking at each node v of G and pairing each incoming with each outgoing edge for v that satisfy the telescoping property. Then it is not hard to see that the nodes of H correspond to length-1 patterns and the edges of H correspond to length-2 patterns.

In order to find the TSG-decomposition of the graph $\hat{G}$ (defined in line 10 of Algorithm 1), we construct the set S of all nodes of H that have no incoming edges. Then the node in S are exactly the roots of the TSGs of the decomposition, and any TSG t can be constructed by doing a depth-first search in H from the node corresponding to the root of t . Since depth-first search takes a linear time to compute [9], the TSG-decomposition can be computed in time linear to the size of H .

5 The generation phase

As mentioned above, the generation phase has two modes, generation of seed traffic and generation of new traffic based on the seed. These stages will be described in the next two subsections.

5.1 The seeding mode

The structure of the seeding algorithm is given in Algorithm 2. We will next discuss some of the steps in more detail.

Algorithm 2 Computing a seed traffic graph

Input: A traffic profile P

Output: A seed graph S and its profile $P(S)$ to be used in the generation phase

```
1: Initialize  $S$  as an empty graph and  $P(S)$  as its
   profile
2: for all  $t \in \{0, w, 2w, \dots, kw = W\}$  do
3:   Determine the weekday  $d$  and the day time  $h$ 
        corresponding to  $t$ 
4:   Pick a number  $Num$  of the TSGs to generate
        consistent with  $num[d, h]$ 
5:   for all  $k \in \{1, \dots, Num\}$  do
6:     Pick a size  $s$  for the next TSG  $T$  using the
         $sz[d, h]$  structure
7:     Initialize  $T$  as the empty tree
8:     Create a root edge  $e$  using the  $ptn[d, h]$ 
        data structures and add it to  $T$ 
9:     for all  $l \in \{2, \dots, s\}$  do
10:      Pick randomly an edge  $h = (t, u)$  from
         $T$  to use as a root for expansion
11:      Create an edge  $f = (u, v)$  using the
         $ptn[d, h]$  structures and add it to  $T$ 
12:    end for  $l$ 
13:  end for  $k$ 
14:  Add  $T$  to graph  $S$ 
15:  Update the  $P(S)$  with the new traffic
16: end for  $t$ 
17: Output  $S$ 
```

The algorithm loops, in lines 2–16, over all time windows of length w , similar to Algorithm 1. (In our implementation, w was chosen to be one hour.) In line 4, the value for Num is picked by sampling a normal distribution with mean $num[d, h]$. Similarly, in line 6, we pick s by sampling a normal distribution with mean $sz[d, h]$.

In line 8, we consider the following cases for defining the root edge e . If $s = 1$, we sample a discrete distribution determined by the frequencies in $ptn(d, h, 0)$, i.e., we pick e with probability proportional to the frequency

of the individual edges in the (d, h) time window. We apply the same procedure if $ptn(d, h, 1)$ has no elements, i.e., if the number s' of TSGs of sizes at least two in the (d, h) time window is zero. Finally, in the case $s \geq 2$ and $s' \geq 1$, we pick e by sampling a discrete distribution determined by the frequencies in $ptn(d, h, 1)$, i.e., the patterns of length 2. Specifically, if the pattern (f, h) is picked, we set $e = f$. The reason we use $ptn(d, h, 1)$ in the case $s \geq 2$ instead of $ptn(d, h, 0)$ is that the vast majority of TSGs are of length 1, hence the latter choice would lead to many cases where e has never been in the same TSG with any other edge and, therefore, in the majority of cases the edges in the TSG rooted at e would not use make use of the patterns from the profile.

Next, if $s > 1$, Algorithm 2 adds one edge at a time to the new TSG T until T 's size becomes s . In line 10, we randomly choose an edge h from the edges in T with probability proportional to the number $n(h)$ of length-2 patterns with first edge h . We also assign a lower non-zero probability for h in the case $n(h) = 0$. In line 11, if there is a length-2 pattern with first edge h , then we choose randomly one such pattern p with probability proportional to the frequency of that pattern relative to other patterns starting with h and define f as the second edge in p . If there is no length-2 pattern with first edge h , then we pick randomly an edge f from $ptn(d, h, 1)$ with first node u , if there is such an edge, with probability proportional to the frequency (multiplicity) of f . Finally, if there is no such edge in $ptn(d, h, 1)$, we pick v randomly from the nodes of G with probability proportional to v 's indegree in G . Finally, in line 15, the profile $P(S)$ is updated by a procedure similar to the one described in Algorithm 2.

5.2 The generation mode

The algorithm for the generation mode is similar to the Algorithm 2 described in the previous section. The only differences are that, instead of using the traffic profile P for the original traffic graph, we use the profile $P(S)$ for the seed graph S , which is updated in line 15 as the time window W slides forward, and instead of the loop in lines 2–16 covering a time interval of W , the length of the loop will be determined of the time period the new traffic is required to cover, which will be an input parameter.

6 Experimental evaluation

For our experiments we used historical ssh network traffic data collected at Los Alamos National Laboratory. The subset used for testing was cleaned (faulty or duplicated data was removed), the packets were stitched together into sessions, and the sessions data converted into

a graph format.

We computed a traffic profile by applying Algorithm 1 on a graph representing traffic for a period W set to 28 days. The resulting profile was used in the generation of a new traffic for a period W . As described above, the generation involves two stages, first generating a seed graph for a period W by using Algorithm 2, and then using the algorithm described in Section 5.2. The seed graph traffic produced in the first stage is ignored and only the traffic produced in the second stage is used in the analysis. The generation process is repeated 10 times with different random number generator seeds and the measured values have been averaged. We compared the indegrees, the outdegrees, the session durations, and the diurnal traffic volume patterns. The results are presented on Figures 2 and 3.

The first plot compares the indegrees of the original and the generated traffic, where the frequency, defined as the number of nodes with indegree d for each number d , in the original and the generated graphs are plotted. Since the generated traffic data is averaged over 10 runs it is smoother, while the original traffic is not averaged and hence the plot is more rugged having for some values of d no nodes with indegree d . For that reason, we plotted a second smoothed version of the original traffic data showing the moving average of the frequency over the last five values, which matches the generated traffic data more closely. The spike at point 200 is because of the fact that all entries with $d > 200$, which are very sparse, have been lumped into single entry with $d = 200$. (This rule has been applied to the other plots as well.)

The second plot of Figure 2 shows a similar comparison for the outdegrees. While the plots are reasonably close, the match is not as good as in the previous case. The reason is most likely due to the way the edges (sessions) are generated. While the target of a new edge is always chosen taking into account the profile information about indegrees, the source of a new edge is chosen based on the ptn pattern information. For instance, if the TSG that needs to be generated consists of the consecutive edges and we have already generated the first edge e_1 , the second edge e_2 is chosen based on information how often the sequence e_1, e_2 occurs in the profile data. Hence, e_2 would be chosen based on information not directly related to the outdegrees. However, e_1 in this example would be constructed using the indegree and outdegree information, hence the generally good (but not perfect) match in the plot.

The last plot of Figure 2 compares the durations. Like the previous plot, there is a generally good match, but also some deviations, especially for low duration values. The likely reason is that, when an edge is created that has

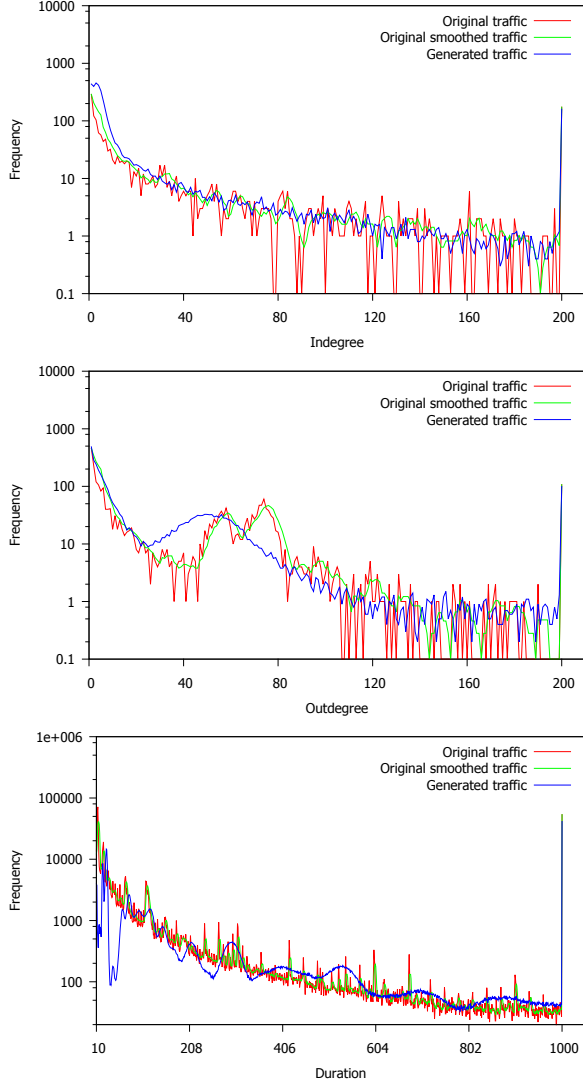


Figure 2: Indegree, outdegree, and duration distributions for the original and generated traffic.

no analogue in the profile (an edge e between two nodes that have never been connected in the traffic used for the profile), the duration is assigned based on the average duration for all profile edges. That increases the number of instances with durations close to the average ones and decreases the number of instances with very small durations.

Figure 3 compares the original and generated traffic with respect to the total traffic volumes for each weekday/hour combination. There is almost a perfect match, which can be explained by the specific way the traffic is generated, namely, that the algorithm generates the same number and the same sizes of the TSGs specified by the traffic profile for each weekday/hour, which uniquely determines the total traffic volume.

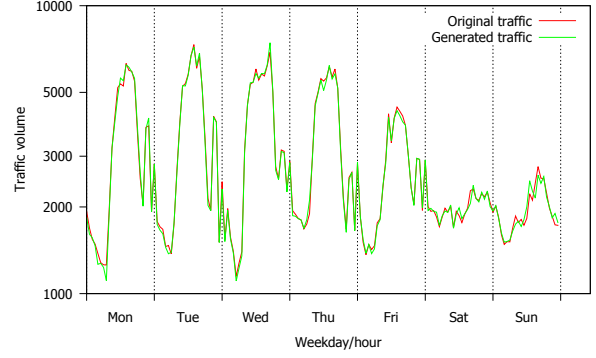


Figure 3: Diurnal patterns for the total volume of the original and generated traffic.

Finally, Table 1 lists the Pearson correlation coefficients for the compared parameters for the original versus the generated and the generated smoothed traffic as discussed above. The table shows that the original and generated traffic variables are very highly correlated, especially in the smoothed case. We can therefore conclude that the generated traffic satisfies the requirements for matching the specified parameters of the original one.

Generated	Indeg	Outdeg	Dur	Volume
Raw	0.84	0.94	0.83	0.99
Smoothed	0.96	0.97	0.93	1.00

Table 1: Correlation between indegrees, outdegrees, durations, and diurnal traffic volume patterns for the original and the generated traffic.

The algorithms are also relatively fast, generating a day’s traffic for roughly 10 seconds on a desktop computer.

7 Conclusion

We described a method for generating ssh traffic that has similar characteristics to a real observed traffic and experimentally analyzed its performance. There are several open problems for future research. One is to generalize the method to capture yearly traffic patterns as the network traffic in days around public holidays (before, during and after), other special events, vacation periods, etc. is expected to be different from the typical patterns during regular working days. It will also be interesting to apply the methodology to communication protocols other than ssh. Finally, the algorithm described here generates only “normal” traffic; for anomaly detection testbeds it is important to have abnormal, or attacker’s traffic. We are planning to address those issues in our future work.

References

- [1] Yu-Chung Cheng, Urs Hölzle, Neal Cardwell, Stefan Savage, and Geoffrey M. Voelker. Monkey see, monkey do: A tool for tcp tracing and replaying. In *USENIX Annual Technical Conference, General Track*, pages 87–98, 2004.
- [2] F. Chung and L. Lu. Connected components in random graphs with given expected degree sequences. *Annals of Combinatorics*, 6(2):125–145, November 2002.
- [3] Hristo Djidjev, Gary Sandine, Curtis Storlie, and Scott Vander Wiel. Graph based statistical analysis of network traffic. In *Proceedings of the Ninth Workshop on Mining and Learning with Graphs*, August 2011.
- [4] R.P. Lippmann, D.J. Fried, I. Graf, J.W. Haines, K.R. Kendall, D. McClung, D. Weber, S.E. Webster, D. Wyschogrod, R.K. Cunningham, and M.A. Zissman. Evaluating intrusion detection systems: the 1998 DARPA off-line intrusion detection evaluation. In *DARPA Information Survivability Conference and Exposition, 2000. DISCEX '00. Proceedings*, volume 2, pages 12–26, 2000.
- [5] Darren Mutz, Giovanni Vigna, and Richard A. Kemmerer. An experience developing an ids stimulator for the black-box testing of network intrusion detection systems. In *ACSAC*, pages 374–383, 2003.
- [6] Josh Neil. *Scan statistics for the online detection of locally anomalous subgraphs*. Phd thesis, University of New Mexico, 2011.
- [7] Joel Sommers, Rhys Alistair Bowden, Brian Eriksson, Paul Barford, Matthew Roughan, and Nick G. Duffield. Efficient network-wide flow record generation. In *INFOCOM*, pages 2363–2371, 2011.
- [8] Joel Sommers, Vinod Yegneswaran, and Paul Barford. A framework for malicious workload generation. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement, IMC '04*, pages 82–87, New York, NY, USA, 2004. ACM.
- [9] Robert Endre Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972.
- [10] Kashi Venkatesh Vishwanath and Amin Vahdat. Realistic and responsive network traffic generation. *SIGCOMM Comput. Commun. Rev.*, 36(4):111–122, August 2006.
- [11] Michele C. Weigle, Prashanth Adurthi, Félix Hernández-Campos, Kevin Jeffay, and F. Donelson Smith. Tmix: a tool for generating realistic tcp application workloads in ns-2. *SIGCOMM Comput. Commun. Rev.*, 36(3):65–76, July 2006.