Behavior-Based Network Traffic Synthesis

Yingbo Song, Salvatore J. Stolfo, and Tony Jebara

***Abstract*—Modern network security research has demonstrated a clear necessity for open sharing of traffic datasets between organizations a need that has so far been superseded by the challenges of removing sensitive content from the data beforehand. Network Data Anonymization is an emerging field dedicated to solving this problem, with a main focus on removal of identifiable artifacts that might pierce privacy, such as usernames and IP addresses. However, recent research has demonstrated that more subtle statistical artifacts, also present, may yield fingerprints that are just as differentiable as the former. This result highlights certain shortcomings in current anonymization frameworks particularly, ignoring the behavioral idiosyncrasies of network protocols, applications, and users. Network traffic synthesis (or simulation) is a closely related complimentary approach which, while more difficult to accurately execute, has the potential for far greater flexibility. This paper leverages the statistical-idiosyncrasies of network behavior to augment anonymization and traffic-synthesis techniques through machine-learning models specifically designed to capture host-level behavior. We present the design of a system that can automatically learn models for network host behavior across time, then use these models to replicate the original behavior, to interpolate across gaps in the original traffic, and demonstrate how to generate new diverse behaviors. Further, we measure the similarity of the synthesized data to the original, providing us with a quantifiable estimate of data fidelity.**

I. Introduction

Open access to quality data is the foundation upon which nearly all modern engineering research disciplines are built upon. The availability of openly distributed and standardized datasets is a prerequisite for scientific measurements in improvements in techniques, and verification and comparisons of existing technologies. In the field of networking research, however, creating publicly releasable datasets is met with many hurdles. First and foremost, as an information-exchange medium, network data contains the personal communications of individuals, and use such data carries privacy concerns. Furthermore, the scale of typical enterprise networks implies a scale of thousands, if not tens of thousands of users, which has in the past made artificial simulation difficult to achieve.

This paper presents a method to synthesize network traffic which can quantifiably mimic real-user behavior. While different types of network traffic generators have been revealed in the past, each serving distinct roles, to the best of our knowledge, ours is the first system which was designed to agnostically mimic user behavior by automatically *learning* behavioral profiles per user, then using these profiles to synthesize new traffic, in a way that is quantifiably similar to the original. In this manner, our system learns to mimic the behaviors of a network as a whole based upon individual members. This work is an extension of our recent results on behavior profiling and anonymization using statistics-preserving anonymity by crowds [1].

Y. Song is with the Department of Computer Science, Columbia University, New York, NY 10027–7003. Phone: +1 646–775–6031, E-mail: yingbo@cs.columbia.edu

S. J. Stolfo is with the Department of Computer Science, Columbia University, New York, NY 10027–7003. Phone: +1 212–939–7080, E-mail: sal@cs.columbia.edu

T. Jebara is with the Department of Computer Science, Columbia University, New York, NY 10027–7003. Phone: +1 212–939–70791, E-mail: jebara@cs.columbia.edu

In the networking and security research community, there is a significant need for publicly available research data. A 2008 survey by Mirkovic showed that out of a total of 144 papers published in Special Interest Group on Data Communication (SIGCOMM) and Internet Measurement Conference (IMC) in 2006 and 2007, 49 of these efforts had utilized network traces in their evaluations, but only 10 had used publicly available datasets [2]. The cause was cited to be a lack of high quality standardized and openly available data. It is because of this need that organizations such as OpenPacket [3], and the more recent U.S. Department of Homeland Security-sponsored PREDICT [4], were recently created – to facilitate the sharing of information between research organizations. However, sharing network data has proven to be difficult, at best. A packet capture of all network traffic, for example, would include web traffic showing which websites users visited, where they transfer files to and from, the locations of their email, banking, and other private accounts, as well as any credentials not protected by encryption. In addition to personal information, the disclosure of network profiles such as vulnerability fingerprints in existing machines, firewall policies, details of existing security services, location of database and other sensitive servers, and network infrastructure in general, can all lead to unintended negative consequences for the releasing party. Network trace anonymization (NDA) is an emerging field that is dedicated to solving this problem. NDA technologies aim to facilitate the exchange of authentic network data by providing the methods by which releasing parties can remove any potentially sensitive artifacts from the data beforehand. Ideally, the anonymized data can then be shared without fear of breeching privacy. Network traffic *synthesis*, or simulation, is a closely related complimentary approach which is more difficult to execute but has the potential for greater flexibility.

In a recent paper, we proposed an anonymization system that utilized machine learning to learn behavioral profiles for network hosts. Then, by inter-mixing traffic amongst similarly behaving members, we derive a pseudo statistical-mixed-net that provided anonymity-by-crowds while preserving the statistical properties within the data [1]. In this paper, we extend this work to demonstrate how, given these behavior profiles, synthetic network data can be generated that mimic the properties of the original traffic. We show how this type of synthetic data can be used for many purposes, such as producing realistic network traffic, reshaping behavior, augmenting anonymization technologies, as well as simulating communications networks. In addition to anonymization, this technology has potential use in areas such as network measurement, and simulation environments such as the DARPA National Cyber Range (NCR) project [5].

The main novelty of our paper is the design of a system that can automatically learn models for network host behavior across time, then use this model replicate that behavior. We show how to manipulate the models to generate diverse behaviors. Further we can measure the similarity of the new data to the original, providing us with a quantifiable estimate of data fidelity.

A. Paper structure

This paper is structured as follows: § II describes background work in network-traffic synthesis and gives an overview of the related works, § III describes the design of our statistical model in detail, § IV provides the algorithms for how to synthesize new network traffic using our models as well as relevant performance measurements, our conclusions and future work are presented in § V and § VI respectively.

II. Background and related works

Research on network data generation has been an interesting topic for some time, for a variety of reasons. Most of these focus on network benchmarking, routing testing, firewall, IDS, and other security related testing. As recently as 2008, great interest has emerged in the network traffic anonymization (NDA) field which studies the intersections between the needs of network measurement and security community balanced with the privacy issues with using data corresponding to human users. This section briefly covers an overview of these related topics and describes our entry into the mix in a novel direction: building systems which automatically models real-user activity and mimics this in new synthesized data.

The 2008 survey by Mirkovic [2] and other similar studies [6], make clear the existence of a deficiency for publicly available large network-traffic datasets. This need has resulted in the introduction of organizations such as OpenPacket [3], and the more recent U.S. Department of Homeland Security-sponsored PREDICT [4] which aim to facilitate the distribution of network data for research purposes. Unlike data used in other disciplines, raw network traffic data often include sensitive information that cannot be released without breech of user privacy. A packet capture of all network traffic, for example, would include web traffic showing which websites users visited, where they transferred files to and from, the locations of their e-mail, banking, and other private accounts, as well as any credentials not protected by encryption. In addition to personal information, the disclosure of network profiles such as vulnerability fingerprints in existing machines, firewall policies, details of existing security services, location of database and other sensitive servers, and network infrastructure in general, can all lead to unintended negative consequences for the releasing party.

NDA technologies are designed to facilitate the exchange of authentic network data by providing methods with which provides can sanitize the data of sensitive information before release. Ideally, the anonymized data can then be shared without fear of breeching privacy. In a recently submitted paper, we proposed an anonymization system that utilized machine learning to learn unique behavioral profiles for each host on a network, then by inter-mixing the traffic amongst similarly behaving hosts, we obtain a pseudo statistical mixed-net which provides anonymity by crowds, while preserving the useful statistical properties of the individuals. In a complimentary approach we study the topic of statistical-behavior-based traffic synthesis for network data generation in this work. By augmenting modified data with synthetic models, we aim to further improve NDA tasks to effectively fuzz the original data and dilute the data of personally identifiable information.

The most well known examples in artificial traffic generation come from the network measurement disciplines. Often, network engineers need to test things such as routing paths, load balancer implementation, firewall policies, and appliances used to maintain the network. In these disciplines the goal is to be able to generate a wide range of data at a fast pace. Breakpoint Systems [7] and Spirent [8] are examples of this role. Traffic generators from these companies can send 40 Gb/s of data at a router, hitting every port, using every protocol, setting every TCP/IP flag, and any enumerations thereof. Phoenix Datacom's Packetstorm Network Simulator will emulate an isolated network for testing purposes [9]. These systems serve distinct network measurement roles.

The second most commonly observed need for traffic generation is for security testing. Some of the previously mentioned systems are capable of generating exploit traffic to test IDS implementations under heavy load. Mostly though, this area is filled with specialist tools that test specific security problems. These tools generate traffic as sort of a a by-product to their main intended use in testing security vulnerabilities, and in that sense they can be considered malicious traffic generators, to an extent. Well known tools in this area are NMap port scanner [10], the Nessus web vulnerability scanner [11], the Metasploit exploitation framework [12], and the list goes on for very long.

In the area of precision network problem diagnosis, specialized packet crafting tools include hping [13] and Nemesis [14]. Ostinato is a GUI-based packet assembler that aims to work like Wireshark [15] in reverse. Network Expect [16] is a powerful tool that allows one to craft and manipulate complicated TCP/IP sessions using a high level scripting language. The difference between these methods and ours is that they serve to serve to generate specific classes of data for specific roles and do not attempt to mimic existing network traffic behavior and synthesize data for the purpose of simulation.

A more closely related area emerged in 2008 when a broad agency announcement was sent out by DARPA describing the goal to create a national testing platform for cyber-warfare technologies. From this emerged the National Cyber Range (NCR) project [5], a multi-million dollar effort to build a system which simulates a pseudo-Internet with thousands of nodes sending and receiving realistic looking traffic. Our proposed method fits well into this category as a technology which can be deployed on an actual network and used to automatically learn host behaviors, then the statistical models are transferred to a separate network where the synthesis algorithms mimic the network hosts that they have learned to emulate for network simulation purposes.

III. Network Traffic Models

Given a packet capture of traffic from an enterprise network, it's very likely that from among the content of the data streams one can glean content signatures that uniquely identify specific individual hosts. For example, by observing web traffic we notice specific usernames being transmitted from a machine to a foreign web server. A collection of such signatures can distinguish one user from another and help form a content-based fingerprint for that user. A related question would then be, is it

possible to obtain a signature for a user based solely on *statistical* information (or "behavior") represented by the meta-data, by tracking rates of packet exchanges, frequencies, and port activities? In a recent paper describing our IMPACT anonymization framework [1] we demonstrated that this is indeed possible. With significant accuracy we can identify users based entirely on this meta-data without any content.

In this prior work our algorithms were used to cluster similarly behaving hosts in an offline transform that simulates the effects of a statistics-preserving anonymous VPN with multiple in and multiple out aggregation points. In this paper, we extend our models to perform data generation for simulation purposes, rather than prediction. Our focus here is to capture a detailed representation of user-activity in a way that can best replicate that behavior in synthetic data. This section describes the mechanics of our statistical model and the following section describes the procedures used to generate new data.

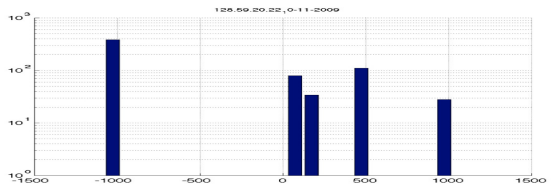
Our algorithm operates on data in the form of Netflow-like statistics. Specifically, we need the following information:

$$\{\text{TIME, SRC, SRCPORT, DST, DSTPORT, \#PKTS}\}$$

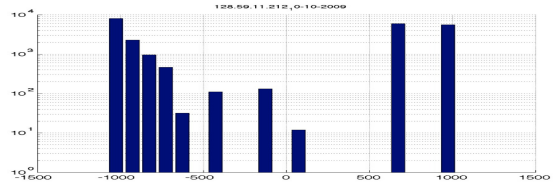
A record for a user's behavior activity would contain hundreds to thousands of these entries, representing behavior throughout the day. Given the port numbers we can estimate which services the user executed. This information is easily extracted from Cisco Netflow records. If the only data is available is in the form of packet captures, then the `softflowd` [17] tool can be used to replay the packets which can then be recaptured in Netflow format using `nfcap` from the `nfdump` [18] tool suite, this effectively converts the packet data into Netflow format.

Measurable traffic data reflect a complex interaction of different inputs throughout the many layers of the TCP/IP protocol stack as well as the state of the machine. A user clicks on a URL on a webpage which causes his browser to dispatch a message to a foreign server in the form of an HTTP request, this request is transmitted in a TCP stream, broken into independent IP datagrams, these datagrams are encapsulated in Ethernet frames and are passed to the local router for transmission. At each of these layers, measurements of quantifiable behavior such as volume and velocity of the packets implicitly take into account the effects of the network stack and the state of the machine in addition to the desired concepts of unique intention or behavior on the part of the user or daemon that is inducing this traffic. Attempting to correlate behavior solely for the inter-arrival times of packets or rates, for example, would incur certain degrees of information loss due to dissimilarities caused by different implementations of the TCP stack, OS version, *etc*.

One must also consider the availability of monitoring methods when designing behavior-based systems. While it is possible to obtain high-level representations of behavior which are independent of the implementation environment, such as HTTP proxy logs which records exactly what the user was attempting to obtain, such high level logging exists for only a handful of services and are diverse in format. In practice, the two most used all-purpose logging methods are packet captures and Netflow captures. It is for these reasons that we have chosen this statistical representation to build our behavior models upon.



(a) User machine port-histogram



(b) Server port-histogram

Fig. 1. Log-scale port histograms. The first half of the x-axis ($x < 0$) represents Received-On ports; the second, Sent-To ports. Notice the distinct behavioral patterns.

A. Statistical representations of behavior

Figure (1) shows an example of the emergence of distinguishable behavior when examining the data at this layer. This plot shows the log-scale plot of the port-traffic histogram of two difference machines on the Columbia network using traffic captured across the span of one day; we see a user machine making requests to various service ports on a foreign machine, and receiving responses back on its ephemeral port. The server machine on the right receives several distinct types of incoming requests and makes three main types of outbound connections: a low-port connection, most likely SSH, an outbound request for a port in the range of 600, and replies on ephemeral ports (in this figure, ephemeral ports are coalesced together on port 1025.

Our model, which will be explained in the following subsections, is time-series analogue of this behavioral snapshot, essentially tracking this behavior of the system *across time*. Given sufficient data sampling, our research results show that very distinct *statistical fingerprints* of host identity emerge. This reflects the natural behavioral idiosyncrasies that lay hidden in plain sight within network data.

B. Protocol-Transition Markov Model

This section describes our model and extensions, we refer the reader to our previous paper for more detailed background and derivation details [1]. The PT-Markov model tracks the dynamics of a system as a transition between specific application services represented by distinguishable protocols (FTP $\rightarrow$ HTTP $\rightarrow$ SMTP, ...). Let s_t represent the "state" of the system at time-step t, or rather the current active service, let $\boldsymbol{x}_t$ represent the packet volume information at that time-step. We then have the following single-step Markov model:

$$p(s, x|\theta) = p(s_1)p(x|s_1)\prod_{t=2}^{T} p(s_t|s_{t-1})p(x_t|s_t). \qquad (1)$$

In the above equation, $p(x|s)$ represents the "emissions" model which describes the output of the system when in that state, in

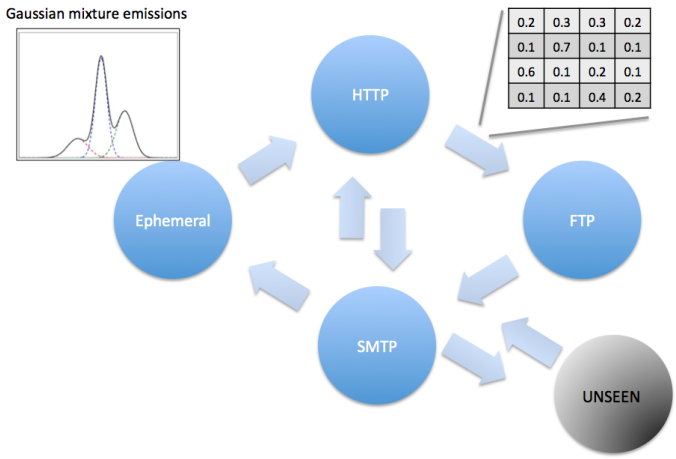


Fig. 2. Protocol-transition Markov model diagram.

our model we use a scalar Gaussian-mixture of the form:

$$p(x|\theta) = \sum_{i=1}^{M} \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left\{-\frac{(x-\mu_i)^2}{2\sigma_i^2}\right\}. \quad (2)$$

A mixture model allows us to implicitly capture an estimate of the different types of traffic exchanges observed under a particular protocol. For example, a distribution on the sizes of websites visited, or distribution on the lengths of emails sent. The above emissions model can be easily estimated using Expectation Maximization (EM). The update rules for EM are as follows, for the current parameter estimate $\theta = \{\pi_1, \mu_1, \sigma_1, ..., \pi_M, \mu_M, \sigma_M\}$:

$$\gamma_t(i,k) = p(s_t = i|x_t, \theta) \quad (3)$$

$$\hat{\mu}_{i,k} = \frac{\sum_{t=1}^{T} \gamma_t(i,k) x_t}{\sum_{t=1}^{T} \gamma_t(i,k)} \quad (4)$$

$$\hat{\sigma}_{i,k} = \frac{\sum_{t=1}^{T} \gamma_t(i,k)(x_t - \mu_i)(x_t - \mu_i)}{\sum_{t=1}^{T} \gamma_t(i,k)} \quad (5)$$

$$\hat{\pi}_i = \frac{\sum_t^T \sum_k \gamma_t(i,k)}{\sum_t^T \sum_{i,k} \gamma_t(i,k)}. \quad (6)$$

θ contains a parameter π which tracks the frequency of each state $\pi_i = p(s_1 = i)$, this parameter is not needed in the likelihood evaluations, however it will be used in the synthesis portion. $\gamma_t(i,k)$ refers the likelihood of s_t being being generated by the k-th sub-model of state i. $\hat{\mu}_i$ and $\hat{\sigma}_i$ in the above equation refers to the updated parameter estimate of θ. Since the value of the state is known, parameter estimation for the state-transition probability table $T(a,b) = p(s_t|s_{t-1})$ consists of simply tracking the transition-frequencies as follows:

$$T(a,b) = \frac{\#(s_i = b, s_{i-1} = a)}{\#(s_i = b, s_{i-1} = a) + \#(s_i \neq b, s_{i-1} = a)}. \quad (7)$$

In the above equation $\#(s_i = b, s_{i-1} = a)$ is a function that counts the number of times state b follows state a in the training data (how many times a user transitions from FTP to HTTP, for example). Putting these together we obtain the full probability distribution $\boldsymbol{s} = \{s_1, ..., s_T\}, \boldsymbol{x} = \{x_1, ..., x_T\}$ and $\Theta = \{\theta_1, ..., \theta_M\}$:

$$p(s_1, x_t) = \sum_{t,k} \gamma_t(i,k) \sum_{k=1}^{K} \mathcal{N}(x_1|\theta_{1,k}) \quad (8)$$

$$p(\mathbf{s}, \mathbf{x}|\Theta) = p(s_1, x_t) \prod_{t=2}^{T} T(s_{t-1}|s_t) \sum_{i=1}^{M} \mathcal{N}(x_t|\theta_{i,k}). \quad (9)$$

At this point, all the defining characters of a behavioral profile is specified and the EM update rules to learn these parameters from the training data has been specified. One last parameter is needed: for each model θ_i we need to track table of source and destination port pairings. Let $b_{i,j} = p(\text{dst} = j|\text{src} = i)$. Table $\mathbf{b}$ is estimated in the same manner as $T(a,b)$. This gives us a final estimate $\theta = \{\mathbf{b}, \pi_1, \mu_1, \sigma_1, ..., \pi_M, \mu_M, \sigma_M\}$.

C. Statistical features and implementation details

One might notice that If we track transition of services based on port values then we theoretically could have a unmanageable $65,536 \times 65,536$ transition matrix. In practice this is not the case due to the design of TCP/IP. There are three types of ports defined in TCP/IP, colloquially referred to as the "well known," "registered," and "ephemeral" ports. Each port type has its own value range. "Well-known" ports extend from port 0 to port 1024 and are used by very common network services such as SSH. The "registered" range extends from 1,025 to 49,151 and is used by third party applications, several Bittorrent clients uses port 6900, for example. Finally all other services with no registered ports uses a randomly selected port from the "ephemeral" range of 49,152 to 65,536. Given this implementation design, two things become apparent: since the "well-known" range tend to experience more stable and dense traffic activity it is best to maintain a one-to-one feature mapping. That is, features relating to SSH should not be combined with features relating to FTP, for example. Secondly, since the "ephemeral" range maintains no correlation between service and port used, we can collapse all activity in this range into a single bin.

The question then remains what to do with the "registered" range. Feature extraction for this range is not well defined for several reasons. First, TCP/IP implementations are not consistent across all platforms and overlapping registration does occur, in addition different operating systems will allocate ports differently based on implementation, version, and on the third party applications themselves. Third, it is not a strict requirement for services to use the ports that they are registered to. Many applications will change ports automatically when necessary, such as when desired ports are already used by other services. Many applications expose this port switching feature to users directly. VNC, for example, increments the port for each active session, starting from port 4900, and allows the user to modify this range. Most Bittorrent clients allow users to change the ports used. Additional unpredictability emerges from uses of NATs, firewalls, tunneling and so on. Treatment of this port range is, therefore, not as well defined as the other two ranges. In our work we decided to use a histogram to represent this range and bin services based on their numerical proximity.

DATASET	NUM HOSTS	BASELINE	MARKOV MODEL	IMPROVEMENT
WEST POINT BORDER	47	2.13%	**60%**	28.2X
NSA CAPTURE	45	2.22%	**37%**	16.7X
LBNL ENTERPRISE	35	2.86%	**78%**	27.3X
COLUMBIA UNIV.	82	1.22%	**76%**	62.3X
GEORGE MASON UNIV.	65	1.54%	**75%**	48.8X
UMICH. MERIT	135	0.74%	**62%**	83.7X

TABLE I
ACCURACY EVALUATIONS ACROSS DIFFERENT DATASETS.

Assuming we have $49,151 - 1,025 = 48,126$ registered ports, we use a 100 bin histogram, we would accumulate activity for all ports between 1025-1124 into the first bin of the histogram, 1125-1224 in the second bin, and so on. This allows us to capture a rough estimate of behavior.

In our experiments, we used a 20-bin histogram, chosen because port-activity is typically very sparse unless some sort of port-scanning activity is occurring. This is because on average, most hosts typically use only a handful of services (SMTP, HTTP, *etc*), mostly in the "well known" range. Sparsity is particularly pronounced in the "registered" range in practice. Using large histograms induces stronger training-data requirements on the machine learning algorithms that train our statistical models, and makes accurate parameter estimation more ill-posed.

When measuring volume, network traffic can fluctuate dramatically for particular hosts even when measuring the same services. This is due to the exchange of protocol-control and data messages, which can induce a large variance in the statistical model and reduce modeling accuracy. To compensate, we use a log-squashing function on the volume feature. Let x represent the volume of activity that a particular service observes for a given session, instead of measuring x in our models we measure $log(x)$.

One last step is to model the distribution of port pairings. The model described in the following subsection tracks the transition of destination ports in network activity in a statistical time-series model to represent services exercised, the type of service is implicitly encapsulated in this representation. In order to faithfully replicate source and destination pairings in subsequent data synthesis tasks we need a representation which explicitly tracks these pairings. Therefore, for each destination port we track the set of associated source ports. When a state is encountered in testing that was not seen in training, for example the appearance of HTTP traffic where only SMTP was observed during training, then no entry for that protocol would exist in the trained model. This can be solved by adding a single dummy state to represent all protocols unseen during training with a fixed value set for both entry- and exit- transition and emission probabilities. This state essentially represents previously unseen traffic.

C.1 Accuracy evaluation

We evaluated the accuracy of our model across a range of datasets. These include the well-known Lawrence Berkley National Labs (LBNL) dataset [19], collections from West Point and National Security Agency (NSA) [20], and University traffic from Columbia University, George Mason University and the University of Michigan/Meritt Networks. For each host, multiple 150-entry segments of Netflow were sampled for testing data, while the rest was used for training. The labels for the testing data is then removed. After training, the model that yielded highest likelihood for a particular test sample produces the label for that sample. Ideally, the correct model products the highest likelihood score, if this is the case the label is considered correct. Since there are no prior techniques that are directly comparable to ours for this particular niche domain of traffic modeling, we use random guessing as a baseline to show the statistical significance of our results. For example, if there are 100 hosts, a baseline would yield a 1% chance of a correct random guess.

Table (I) shows the performance of our model. Our model consistently achieves significantly higher performance than the baseline. Our model requires sessions on the order of minutes to hours, and works very well even in cases of very sparse training data, on the order of 20 minutes to an hour. In this setting, our models can attribute unlabeled traffic to the the true host with average accuracies report in the table. This quantifies the degree to which our models are correctly capturing statistical representations for the idiosyncrasies of network behavior. The low-performance on the NSA dataset can be attributed largely to session-reconstruction errors. Poor reconstruction from packet to Netflow statistics can lead to inaccurate records on session duration, size, *etc.*, and this inaccuracy influenced poorer performance on our model.

IV. SYNTHESIS

This section describes our algorithm for synthesizing both Netflow and packet traffic from our statistical model. We describe the technical details of our algorithm and provide full pseudocode for our functions. First, we first describe an algorithm to generate full Netflow-like statistics in § A. In § B we demonstrate a quantifiable improvement in the fidelity of this generated data by comparing our results to data generated using randomly spliced segments of traffic from similar hosts. We show that behavior interpolation is possible using a switching model in § C. An algorithm for data interpolation by filling in missing portions of traffic using a least-squares method is presented in § D. We show to how to convert these Netflow statistics to PCAP in E. A time-driven synthesis model which can replicate a user's activity throughout an entire day is presented in § F. Finally, in § G, we show implementation costs in terms of memory requirements and runtimes for our algorithm.

DATASET	GENUINE	SYNTHETIC	RANDOM SAMPLED
WEST POINT BORDER	-4.1445E+10	-6.6203E+10	-8.3211E+10
NSA CAPTURE	-5.2935E+10	-6.5322E+10	-6.6581E+10
LBNL ENTERPRISE	-3.9719E+10	-6.8987E+10	-8.8169E+10
COLUMBIA UNIV.	-3.0717E+10	-3.1106E+10	-5.5863E+10
GEORGE MASON UNIV.	-1.3528E+10	-3.5596E+10	-3.7741E+10
UMICH. MERIT	-2.0368E+10	-4.1504E+10	-5.2505E+10

TABLE II

LOG-LIKELIHOOD VALUES OF GENUINE, SYNTHETIC, AND RANDOMLY SAMPLED DATA. AVERAGE OF 10 TRIALS REPORTED.

A. Synthesizing Netflow

A few simple mathematical notations needs to be introduced before describing the system. Let $s \sim \mathcal{M}(p_1, p_2, \ldots, p_k)$ denote a random draw from a multinomial distribution specified by the normalized ratios parameter $\mathbf{p} = \{p_1, p_2, \ldots, p_k\}$. Meaning, if we had an alphabet of k characters with the respective probabilities of appearance $\mathbf{p}$ where $\sum_i p_i = 1$, as the number of draws grows the normalized proportion of selected values approaches $\mathbf{p}$. Let $x \sim \mathcal{N}(\theta_i)$ denote sampling a scalar quantity from the Gaussian mixture denoted by $\theta_i = \{\mu_{i,1}, \sigma_{i,1}, \ldots, \mu_{i,m}, \sigma_{i,m}\}$ for mixture size m. The pseudo code for our synthesis algorithm is given as follows:

SYNTHESIZE-NETFLOW(θ, n)

return FEATURES-TO-DATA(SYNTHESIZE-STATS(θ, n))

SYNTHESIZE-STATS(θ, n)

$\triangleright$ Use the frequency estimate to set the initial state
$d_0 \sim \mathcal{M}(\pi_1, \pi, \ldots, \pi_k)$
$\triangleright$ Src/Dst port model used to estimate the pairing
$s_0 \sim \mathcal{M}(b_{d_0,1}, b_{d_0,2}, \ldots)$
$x_0 \sim \mathcal{N}(\theta_{s_0})$ $\triangleright$ Sampling the volume
$\triangleright$ The rest is generated by induction
for $t \leftarrow 1$ **to** $n-1$:
  do $d_t \sim \mathcal{M}(a_{d_{t-1},1}, a_{d_{t-1},2}, \ldots, a_{d_{t-1},k})$
    $s_t \sim \mathcal{M}(b_{d_t,1}, b_{d_t,2}, \ldots)$
    $x_i \sim \mathcal{N}(\theta_{s_t})$
return $[\mathbf{s}, \mathbf{d}, \mathbf{x}]$

The synthesis works by using the service (port) transition probabilities learned during training as parameters in a multinomial distribution, which is then used to sample the most likely subsequent service. This gives us a maximum-likelihood transition between destination ports in a series. For each destination element, recall that we have a separate model for the ratio of source ports in the $\mathbf{b}$ parameter. This too, is a ratio-based parameter and thus can be used to generate a new sample from a multinomial distribution. Given the strong correlation between source and destination ports, second order modeling for source and destination pairings is redundant, that is, we do not need to explicitly model $p(s_i, d_i)$ as opposed to our current $p(s_i|d_i)$. Finally volume information x_i is drawn from the Gaussian mixture parameter which is unique to each service model i. Function SYNTHESIZE-STATS simulates traffic patterns using our feature representation. This result must then be converted back to the normal data-space by the following function.

FEATURES-TO-DATA$(\mathbf{s}, \mathbf{d}, \mathbf{x})$

$[\mathbf{s}', \mathbf{d}', \mathbf{x}'] \leftarrow \emptyset$
$n \leftarrow$ LENGTH$(\mathbf{s})$
for $t \leftarrow 1$ **to** n :
  do $s'_t \leftarrow$ BIN-TO-PORTS(s_t)
    $d'_t \leftarrow$ BIN-TO-PORTS(d_t)
    $x'_t \leftarrow \exp(x_t)$
return $[\mathbf{s}', \mathbf{d}', \mathbf{x}']$

BIN-TO-PORTS is function that reverses the feature-extraction transformation performed on the dataset during training and remaps the data from features back to their normal representation. For example, $d_i = 1045$ then d_i gets remapped to a random ephemeral value between the range of $[49152, ..., 65536]$.

BIN-TO-PORTS(x, h)

if $x < 1026$: $\triangleright$ Well known ports
  then return x
elseif $x > (1025 + h)$: $\triangleright$ Ephemeral ports
  then return RAND$(1, \ldots, 16384) + 49151$
$z \leftarrow (49152 - 1025)/h$ $\triangleright$ Registered ports
return $1025 + (x - 1025) \times h +$ RAND$(1, \ldots, h)$

B. Quantifying Similarity

We quantify the fidelity of the synthesized data by evaluating the log-likelihood of the generated data over the trained models. A faithful reproduction would yield a higher log-likelihood score than a poorly recreated sequence. The likelihood score is calculated using Equation (9) in the previous section. $\log(p(\mathbf{s}, \mathbf{x}|\Theta))$ is used to avoid underflow errors. We compare the log-likelihood scores of synthetic data with the original (genuine) data as well as data piece together by randomly selecting subsets of other hosts's original traffic. The latter case, represented in the table by "Random" is meant to represent a naive way of simulating data. We chose to randomly sample real data vs generating completely random data as this gives us a more challenging performance baseline to compare against given that we are using considering real traffic. Table (II) shows this likelihood comparison. All results are average of 10 randomized trials. As expected, our synthesized data consistently exhibit higher loglikelihood than the randomly sampled data, but lower than the genuine set, which represents an upper bound. This means that our method of synthesizing data is more accurate than if one were to randomly piece together sessions of *actual* traffic from other hosts in the same dataset.

C. Interpolation and Behavior Shaping

Behavior interpolation can be achieved by using a switched-Markov-model. In this case, multiple behavioral models are connected using a new external state variable. During the chain of data synthesis, this variable switches between multiple models probabilistically under a user-specific distribution. A simple method is to simple provide a ratio parameter and sample from a multinomial, as done previously. SYNTHESIZE-MIXED-NETFLOW provides the pseudocode for this procedure.

SYNTHESIZE-MIXED-NETFLOW$(\Theta, n, \mathbf{p}, c)$

$[\mathbf{s}, \mathbf{d}, \mathbf{x}] \leftarrow \emptyset$
for $t \leftarrow 1$ **to** $\lfloor n/c \rfloor$:
  do $i \sim \mathcal{M}(\mathbf{p})$
    $[\mathbf{s}_t, \mathbf{d}_t, \mathbf{x}_t] \leftarrow$ SYNTHESIZE-NETFLOW(θ_i, c)
    $[\mathbf{s}, \mathbf{d}, \mathbf{x}] \leftarrow$ APPEND$([\mathbf{s}_t, \mathbf{d}_t, \mathbf{x}_t])$
    $t \leftarrow t + 1$
return $[\mathbf{s}, \mathbf{d}, \mathbf{x}]$

Data synthesis using switched-models involves generating blocks of traffic of length c for each model and combining them. The result is that individual sections of the data will be statistically similar to different models. In practice, c can be randomized at each iteration within the loop if an exact final length is not required. One might consider why c should not be set as 1. The reason for this is that the PT-Markov model measures *transitional* probability. Therefore, if $c = 1$, in the worst case where we have two completely different behavior models, we can have a thrashing scenario where the context occurs after each generated entry. This would, in theory, yield a result that is dissimilar to both of the initial models. This thrashing effect is mitigated if two models are similar to each other, meaning they share a certain amount of overlap in ports used, thus allowing the models to transition between each other more smoothly. In practice, c can be adjusted based on the model similarities.

To highly another aspect of our model that we have developed, we choose to use a *visual* similarity-measurement technique to quantify the the fidelity of the synthesized data in this interpolation setting. We confirm similarity by automatically plotting new points in a 2-D plane where each 2-D point correspond to a synthesized data sample, and their distances in this plane corresponds to their distances in their original form. This is known as "embedding" and we have developed techniques for network data embedding in our prior work [1]. We began with two classes of behavior, one modeled on a web server (denoted by "web") and the other modeled on a general-computation server (denoted by "sos"). A switched model is used to synthesize samples of their interpolated behavior which we labeled the"S" class. Visually, we can confirm that the synthesized data indeed falls between these two distinct classes of behavior. The exact method which we used to map this abstract behavior manifold is known as kernel principle component analysis (KPCA). The kernel used in this case is a variation of the Probability Product kernel [21] derived specifically for the PT-Markov model. For more details we refer the reader to our other paper which describes this work in more detail [1].

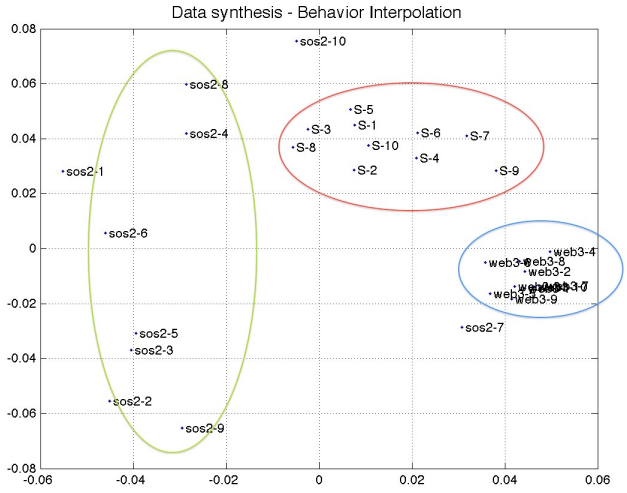


Fig. 3. Behavior interpolation: Two clusters of behavior profiles are embedded ("sos" and "web") along with a new class of behavior ("S") that interpolates between these two models.

D. Behavior Regression

In addition to behavior interpolation and shaping, we can use these models to perform data regression to fill-in missing segments of data. Ideally, we would like the synthesized data to match with the actual data along the boundaries so that transition probabilities are properly maximized. While this is not so difficult if we use a simple one-step Markov model as the basic setting of our model uses, problems would arise in the future if we were to extend our model and use a multi-step Markov dependency assumptions where we have $p(s_t|s_{t-1}, s_{t-2}, \ldots, s_{t-k})$. In this case aligning the boundaries becomes more difficult. Given this we propose a more general solution that can work for any variation on the behavior models. An efficient randomized algorithm is to simply generate many solutions and pick the one that yields the highest likelihood given the choice of probability model $p(\mathbf{s}, \mathbf{x}|\Theta)$. This takes the form of the following algorithm:

NETFLOW-REGRESSION$(\theta, \mathbf{s}, \mathbf{d}, \mathbf{x}, c)$

$n \leftarrow$ SOME LARGE NUMBER
$D \leftarrow [\infty, \infty, \ldots, \infty]$ of length n
$[\mathbf{s}', \mathbf{d}', \mathbf{x}'] \leftarrow$ SYNTHESIZE-NETFLOW(θ, n)
for $t \leftarrow 4$ **to** $n - c - 4$:
  do $D_t = \sum_{i=t-3}^{t} ||d_i - d'_i|| + \sum_{i=t+c}^{t+c+3} ||d_i - d'_i|| \ldots$
    $+ \sum_{i=t-3}^{t} ||s_i - s'_i|| + \sum_{i=t+c}^{t+c+3} ||s_i - s'_i||$
$j \leftarrow$ FIND-INDEX-OF(MIN(D))
$[\mathbf{s}*, \mathbf{d}*, \mathbf{x}*] \leftarrow [s'_{j,\ldots,j+c-1}, d'_{j,\ldots,j+c-1}, x'_{j,\ldots,j+c-1}]$
return $[\mathbf{s}*, \mathbf{d}*, \mathbf{x}*]$

Here a long sequence is synthesized and a sweep is performed to find the indices where the boundaries are most similarly aligned between the original data and the synthetic data, as measured using a simple Euclidean norm. The values between the boundaries within the synthetic data are then taken to patch the missing gap of size c within the original.

26910	4765	46	26910	4765	46
25156	62631	116	25156	62631	116
80	5732	184	**80**	**7178**	**143**
443	63653	52	**443**	**45850**	**222**
80	52282	419	**80**	**31877**	**124**
25443	60829	46	**443**	**34456**	**72**
443	45368	92	**25514**	**4390**	**133**
25186	59341	95	**443**	**59685**	**124**
27473	50041	92	**26136**	**58498**	**117**
80	64955	116	**26772**	**60792**	**279**
25161	4145	819	25161	4145	819

Fig. 4. Filling in missing data: (Left) Shaded entries represents missing data. (Right) Synthesized entries.

Figure (4) provides an example of this regression method. Here, we have pre-trained a model for a particular host. We took a previously unseen segment of traffic from this host and removed a portion of it, this is represented by the shaded values on the left side of the figure. On the right side of the figure, denoted in bold font, we show the sequence of entries generated using our regression technique. Visually, we can confirm the similarity between the real and synthesized data.

E. Netflow to PCAP

Finally, given that many network research tasks require packet capture data, we approach the problem of translating the Netflow-like statistical data that we have generated into packet format. This portion of our work is still under development as synthesizing authentic-looking packet content to replicate the presence of network jitter and common background noise is quite challenging. Our current method loops through the list of entries in the Netflow data and for each session we use the `text2pcap` tool to synthesize a packet session. Subsequently, we replace the source and destination IP addresses (and associated ethernet headers) which can be recorded in a table, separately. These sessions are merged to form the final pcap file.

STATS-TO-PCAP($\mathbf{s}, \mathbf{d}, \mathbf{x}$)
1 PCAP-FILE $\leftarrow \emptyset$
2 $n \leftarrow$ LENGTH($\mathbf{s}$)
3 **for** $t \leftarrow 1$ **to** n :
4 **do** DATA $\leftarrow$ READ-BYTES(/DEV/URANDOM, x_t)
5 SESSION $\leftarrow$ TEXT2PCAP(DATA)
6 SESSION[SRC IP] $\leftarrow s_t$
7 SESSION[DST IP] $\leftarrow d_t$
8 APPEND(PCAP-FILE, SESSION)
9 **return**

Typically, publicly released datasets contain only packet headers and no data content. Generating of content is beyond the scope of this work, which focuses on the network-centric behavioral profiles. Currently, we are reading random data from `/dev/urandom` for the packet contents. Figure (5) shows an example of a packet trace being synthesized; a single session is shown. The current limitation of this method is that the generated traffic is fairly "clean" and network jitters are not reproduced, at the moment. For example, on a normal network is it very common to see such things as bogon traffic, IP fragmentation, duplicate packets, malformed headers, *etc*. These eccentricities are difficult to replicate faithfully given that they are content oriented. Synthesis of these objects remains the subject of our ongoing research.

F. Time-driven synthesis

With the ability to accurately replicate behavior, we complete our synthesis methodology by incorporating a timing mechanism to mimic a particular host's behavior throughout the day. Given that behavior can be replicated, the only missing feature is to model the time of day when a host is active as well as the volume of activity for that time-period. This level of granularity requires far more training data than the prior algorithms since we need to observe a host's behavior throughout the day, for several days, in order to obtain an accurate measurement to create a distribution based on active times. If such data is not available then a randomized algorithm may be used to generate the data sporadically throughout the day. However, if accurate real training data is available, then we can model the true distribution.

We do this by generating a histogram of the user's activity throughout the day and breaking each day into 288 individual five-minute time slices from midnight to midnight. For each time slice, we measure the aggregate volume of data sent to and received by a particular host. This gives us a statistical sample $\mathbf{y} \in \mathbb{R}^{1\times 288}$ for a particular day. With a set of m samples $Y = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_m\}$ we can use a set of Gaussian distributions to track behavior conditioned on time. For each time-slice of activity t we have:

$$\mu_t = \frac{1}{m}\sum_i^m y_{i,t} \tag{10}$$

$$\sigma_t = \frac{1}{m}\sum_i^m (y_{i,t} - \mu)^2. \tag{11}$$

This implicitly assumes independency between the time slices throughout the day. At the five-minute level, this assumption is not too restrictive. Representing this timing model using $\tau = \{\mu_1, \mu_2, ..\mu_{288}, \sigma_1, \sigma_2, ..\sigma_{288}\}$, the pseudo-code for the time-depend packet synthesis is as follows:

SYNTHESIZE-DAY(θ, τ)
1 **for** $i \leftarrow 1$ **to** 288 :
2 **do** $v_t \sim \mathcal{N}(\tau_t)$
3 **if** $v_t < \epsilon$: $\quad \triangleright$ ϵ is a threshold for activation
4 **then continue**
5 $n \leftarrow \left\lfloor v_t / \frac{1}{k}\sum_i^k \mu_i \right\rfloor$
6 $F_i \leftarrow$ STATS-TO-PCAP(SYNTHESIZE-STATS(θ, n))
7 $\triangleright$ Merge all PCAP files into $F*$
8 $F* \leftarrow$ MERGE-PCAP($F_1, F_2, \ldots, F_{288}$)
9 **return**

In our experiments our models have yielded interesting results, such as replicating port-scanners. And while using our models to cluster existing data, we have discovered clusters of machines which were part of botnets. The full potential for this niche area of behavior-based network traffic synthesis is yet unexplored and is the subject of our on-going work.

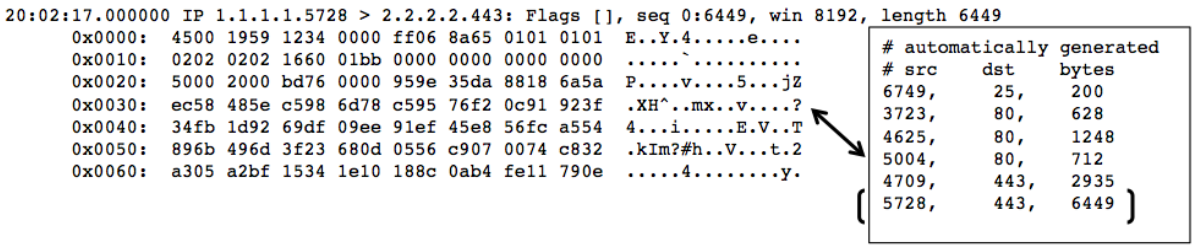


Fig. 5. Translation between statistical representation to PCAP.

G. Memory cost and runtimes

The memory footprint of our models are small when implemented using efficient underlying data structures. The largest part of the model is the transition-probabilities table. This size is upper bounded by the size of the list of potential port features. In our implementation, we had 1025 well known ports, 20 registered-ports bins, 1 ephemeral-port bin, which yields a 1046×1046 transition table. If double precision floating point storage is used for all values then we have a roughly 8Mb maximum storage requirement per model. However, the transition-probabilities table is typically very sparse, because most hosts exercises only a small subset of potential network protocols, unless a host is experiencing some sort of port-scanning activity. In practice, for a typical host which uses a dozen network services, the storage requirement using sparse-matrix implementation is roughly 12Kb per host. This means that on modern computing platforms it is easily possible to compute profiles for networks containing tens of thousands of hosts and keep all parameters in memory simultaneously.

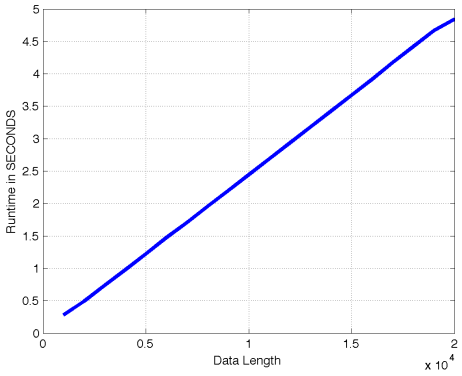


Fig. 6. SYNTHESIZE-NETFLOW maintain $O(n)$ growth. Large traffic set with 20,000 session entries took less than five seconds to synthesize.

Figure (6) shows the runtime of our synthesis algorithm. As expected, our algorithm grows linearly in $O(n)$. Even in SYNTHESIZE-DAY where two loop layers exist, the outer loop is upper bounded by a constant, therefore the entire system still grows in $O(n)$. The experiment shown in Fig. (6) shows runtimes for synthesizing the behavior of a host which had used 16 network services. The figure shows that synthesizing a relatively large 20,000 element Netflow record required less than five seconds. Our computation was performed on a 64-bit 2.66Ghz processor with six cores. We believe our relatively un-optimized research-oriented code can be improved in production settings and significantly higher performance can be achieved.

V. CONCLUSION

In this paper we have described a model for representing network behavior at the host level, for purposes of network traffic simulation. We showed how to extend these models and use them to synthesize realistic looking traffic that mimic the original hosts. We demonstrated how to modify these models to manipulate the resulting synthetic data, by adjusting the parameters, and interpolating between different models to simulate adjustable new behavior. We further showed how to use our models to perform data regression, to fill in "missing" gaps of traffic. Methods to quantify the fidelity of these algorithms are presented, and backed up by experiments which demonstrate favorable performance of our algorithm. To the best of our knowledge, we have presented the first system that can deployed to a foreign network environment, automatically learn the behavior of hosts on the network, then synthesize new traffic which simulates this behavior on a separate network.

VI. FUTURE WORKS

While our technology has demonstrated success in mimicking host-level behavior on a per-machine basis, additional research needs to be done in modeling the complex interconnectivity patterns existing in modern network systems. The unique behavior of a specific subgroup of hosts is particularly interesting, in order to discover trends in communication patterns and changes over time, modeling networks from the top-down. Our future works include using learning techniques to model the interactivity patterns of networks. Drawing results from graph theory, we hope to be able to predict behavior for groups of hosts, much in the same way that we have done for single hosts. Our research in applying this technology to network source-anonymization [1] is also on-going. Ultimately, our goals extend to synthesizing realistic packet *content* as well.

ACKNOWLEDGMENTS

We thank Prof. Angelos Stavrou and Brian Schulte at GMU, and Kyle Creyts at Merit Network for providing us with valuable data (UMich. dataset.) This research was sponsored by Department of Homeland Security, SPAWAR Contract No. N66001-09-C-0080, Privacy Preserving Sharing of Network Trace Data (PPSNTD) Program and a DURIP Instrumentation grant from AFOSR (FA 99500910389).

REFERENCES

[1] Y. Song, S. J. Stolfo, and T. Jebara, "Markov models for network-behavior modeling and anonymization," Columbia University, Technical Report cucs-029-11, June 2011.

[2] J. Mirkovic, "Privacy-safe network trace sharing via secure queries," in *Proceedings of the 1st ACM workshop on Network data anonymization*, 2008.

[3] R. Bejtlich, J. Cummings, and S. Parsell, "Openpacket.org: a centralized repository of network traffic traces for researchers," 2011. [Online]. Available: https://www.openpacket.org

[4] (2011) PREDICT: The Protected Repository for the Defense of Infrastructure Against Cyber Threats. http://www.predict.org. [Online]. Available: http://www.predict.org

[5] DARPA, "National Cyber Range (NCR) program," DARPA-BAA-08-43, 2008.

[6] M. Alllman and V. Paxson, "Issues and etiquette concerning use of shared measurement data," in *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, ser. IMC '07. New York, NY, USA: ACM, 2007, pp. 135–140. [Online]. Available: http://doi.acm.org/10.1145/1298306.1298327

[7] BreakingPoint Systems, "Breakingpoint network traffic generation systems," 2011. [Online]. Available: http://www.breakingpointsystems.com/

[8] Spirent, "Spirent hypermetrics 40/100g module," 2011. [Online]. Available: http://www.spirent.com/

[9] Phoenix Datacom, "Packetstorm Network Simulator," 2011. [Online]. Available: http://www.phoenixdatacom.com/packetstorm.html

[10] G. Lyon, "Nmap - free security scanner for network exploitation & security audits," 2011. [Online]. Available: http://nmap.org

[11] Tenable Security, "Nessus: The network vulnerability scanner," 2011. [Online]. Available: http://www.tenable.com

[12] Metasploit, "Metasploit Project," 2010, http://www.metasploit.com.

[13] S. Sanfilippo, "hping: active network security tool," 2006. [Online]. Available: http://www.hping.org/

[14] J. Nathan, "Nemesis: Packet injection tool suite," 2011 2011. [Online]. Available: http://nemesis.sourceforge.net/

[15] W. Foundation, "wireshark: network protocol analyzer," 2011. [Online]. Available: http://www.wireshark.org/

[16] E. Paris, "Network expect," 2010. [Online]. Available: http://netexpect.org/

[17] D. Miller, "softflowd - fast software netflow probe," March 2011. [Online]. Available: http://www.mindrot.org/projects/softflowd/

[18] P. Haag, "nfdump: Netflow processing tools," Sourceforge, 2011. [Online]. Available: http://sourceforge.net/projects/nfdump/

[19] M. Allman, M. Bennett, M. Casado, S. Crosby, J. Lee, B. Nechaev, R. Pang, V. Paxson, and B. Tierney, "LBNL/ICSI Enterprise Tracing Project," 2005. [Online]. Available: http://www.icir.org/enterprise-tracing/

[20] United States Military Academy, "ITOC CDX Research Dataset," 2009. [Online]. Available: http://www.itoc.usma.edu/research/dataset/

[21] T. Jebara, R. Kondor, and A. Howard, "Probability product kernels," *Journal of Machine Learning Research*, vol. 5, pp. 819–844, 2004.