Markov Models for Network-Behavior Modeling and Anonymization

Yingbo Song
Intrusion Detection Sys. Lab
Dept. of Computer Science
Columbia University
yingbo@cs.columbia.edu

Salvatore J. Stolfo
Intrusion Detection Sys. Lab
Dept. of Computer Science
Columbia University
sal@cs.columbia.edu

Tony Jebara
Machine Learning Lab
Dept. of Computer Science
Columbia University
jebara@cs.columbia.edu

ABSTRACT

Modern network security research has demonstrated a clear need for open sharing of traffic datasets between organizations, a need that has so far been superseded by the challenge of removing sensitive content beforehand. Network Data Anonymization (NDA) is emerging as a field dedicated to this problem, with its main direction focusing 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. Recent anonymization results have shown that the extent to which utility and privacy can be obtained is mainly a function of the information in the data that one is aware and not aware of. This paper leverages the predictability of network behavior in our favor to augment existing frameworks through a new machine-learning-driven anonymization technique. Our approach uses the substitution of individual identities with group identities where members are divided based on behavioral similarities, essentially providing anonymity-by-crowds in a statistical mix-net. We derive time-series models for network traffic behavior which quantifiably models the discriminative features of network "behavior" and introduce a kernel-based framework for anonymity which fits together naturally with network-data modeling.

Keywords

Network behavior, Anonymity, Markov, Time-series, Kernel

1. INTRODUCTION

Modern network security research has demonstrated a clear necessity for the open sharing of large network traffic datasets between research organizations. Many security-related research fields, such as detecting exploits, DDoS attacks, or worm outbreaks, would benefit greatly if researchers had the ability to easily correlate information between several different resources, thus allowing them to extend their scope beyond their own organization's networks. This would encourage both collaboration, and facilitate confirmation of research results. To date, however, sharing of large-scale network traffic datasets, such as packet or Netflow captures, has been relatively limited in scale. This is due primarily to the fact that such datasets often contain sensitive content, including but not limited to, personally identifiable information of third parties not directly involved in the research – where the inadvertent release of such information may cause damage to the releasing entity. As a result, researchers often evaluate their technologies solely on their own organization's own traffic, making direct comparisons of related systems difficult to achieve.

Network Data Anonymization (NDA) is emerging as a field that is dedicated to solving this problem [1]. The predominant direction in NDA is content removal and masking, which includes deletion of packet payloads and masking of headers; such as the removal of flags, and one-way transforms on IP addresses. The most well known tool in this area, for example, is `tcpmkpub` [21], which is a policy-driven framework for utilizing a range of such transformations. Tools such as these facilitate the removal of human-identified signatures which might fingerprint users, hosts, or services which should otherwise remain anonymous.

However, recent research has demonstrated that beyond superficially observable datums such as IP addresses, more subtle *statistical* artifacts are also present in these traces which may yield fingerprints that are just as differentiable as the former. Further, statistical models trained on these artifacts may be used to breach confidentiality. For example, previous work with hidden Markov models (HMM) trained on packet timings for network protocols demonstrate that, even if the traces are encrypted, HMMs were still able to reliably isolate these protocols from the encrypted dataset [29]. This examples highlight certain shortcomings in current anonymization frameworks; particularly, in ignoring the idiosyncrasies of network protocols, applications, and users. As the field of network traffic anonymization progresses, it is certain that behavioral fingerprints should be taken into account.

1.1 Motivation

This paper aims to demonstrate that in a state of uncertainty, it is possible to leverage behavioral idiosyncrasies in our favor, by using machine-learning-driven methods to conceal both the known and the unknown data. Through the substitution of individual identities with group identities, each host is assigned to a particular group based on the similarity of their own behavior to those of other group members. In addition, perturbation of user-behavior is used, and synthetic data drawn from learned behavioral-profiles are inserted into the dataset. We draw from the well-known principle of mixed-nets to provide anonymity by crowds. The intermixing (or "clustering") of these identities effectively anonymizes each member of that cluster (providing k-anonymity) while simultaneously preserv-

ing, in the aggregate, the statistics that characterize the members of that group. "Mixing" data in a group identity takes the form of source aggregation; for example, assigning a single IP address to all members of a group. Normally, any aggregation of statistics naturally causes data-loss, as this translates to capturing a coarser snapshot of the data. However, using statistical-modeling theory from machine learning allows us to drive these transformations in a way such that information-dilution is minimized. As this process is independent of other anonymization transforms, it is compatible with existing frameworks [21, 12] and can be considered as another layer in existing anonymization chains.

Importantly, our research is founded on a *kernel*-based[1] framework, using graph-cutting methods for clustering, and closely related low-dimensional embedding techniques for visualization which extend directly from this work. This approach, as we demonstrate, is a natural fit for the network-traffic domain. By jointly leveraging both Network Security and Machine Learning using this time-series kernel-based framework, we explore a new field of potential opportunities at the intersection of these domains.

1.2 Novel contributions

This paper presents the following novel contributions: 1) We present new feature extraction algorithms that effectively quantifies host-behavior based on network traffic. 2) We derive new time-series models for network behavior as well as kernel similarity functions for these models. 3) We present a clustering and visualization framework based on graph-cutting algorithms and time-series kernels, to identify groups of similarly-behaving hosts. 4) We use these techniques to derive a new anonymization methodology and demonstrate a framework for integration of our technology into existing anonymization platforms.

1.3 Outline

This paper is outlined as follows: § 2 describes background work in packet trace anonymization and gives an overview of the most relevant related works, § 3 covers network behavior modeling in vector and time-series domains. Clustering using time-series kernels and Visualization are found in § 4, and finally, our framework for network trace anonymization is presented in § 5.

1.3.1 Definitions and notations

In this paper, math variables in lower-case bold font such as $\mathbf{x}$ and $\mathbf{y}$ denote column vectors. Non-bold lower-case variables, such as d, denote scalar values. We use $\mathbf{x}_i$ to denote the i^{th} vector of a set of vectors and use $\mathbf{x}(j)$ to denote the j^{th} scalar component of that vector. Matrices are denoted using bold, capital letters such as $\mathbf{A}$. Similarly, the $(i,j)^{th}$ component of a matrix is denoted $\mathbf{X}(i,j)$. $\mathbf{x}^\top$ denotes the transpose of $\mathbf{x}$. "Kernel" refers to the mathematical inner product of vectors under a feature-space mapping and, unless otherwise stated, does not represent the Operating Systems concept.

2. BACKGROUND AND RELATED WORKS

2.1 The necessity of network-data sharing

In the year 2004, a hacker known as "Stakkato" broke into Teragrid, a large world-distributed research computing platform [27]. The attacks spanned a total of 19 months, and successfully infiltrated thousands of university, corporate, and military machines, in both the US and Europe. Forensics revealed that the attacks were quick but consistent, however, collaborative mitigation and defense efforts were hampered by the inability of the individual organizations to freely share data amongst themselves. This was due to the fact that, among the fingerprints and clues that forensic experts might want to extract from such network traffic, the data contained private sensitive information which could not be released, which the owners of such data could not easily remove.

Beyond forensics, it is a common goal of all scientific communities to share data, for purposes of cross-environment testing of proposed algorithms, as well as results verification and reproduction. It is because of this need that organizations such as OpenPacket [4], and the more recent U.S. Department of Homeland Security-sponsored PREDICT [5], were recently created. However, unlike other disciplines, raw network traffic data often include sensitive information. 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.

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 had utilized network traces in their evaluations, but only 10 had used publicly available datasets [18]. This result, along with other published opinions of a similar nature [6], reflect a deficiency in the current network and security research fields of publicly available large traffic-capture datasets.

2.2 Pitfalls of synthetic datasets

It is also pertinent to consider why synthetic data is not sufficient as a substitute. The reason for this is that generating realistic and useful data is, at best, an art that has yet to be perfected. Achieving realism for certain research purposes is a lofty goal that can sometimes fail. One example is the 1998 DARPA KDD dataset which was synthetically generated for IDS testing. After several papers and results were published, it was discovered that all benign packets in this dataset had a TTL entry of 127 or 254, whereas all malicious packets had a TTL of 126 or 253, leading researchers to declare this dataset as "fundamentally broken," and subsequently all submitted papers using this dataset as the basis of their work were rejected [9]. While this is an extreme example such problems, it nevertheless highlights a pitfall in this particular approach.

Very powerful enterprise-level generators do exist; produced by companies such as BreakingPoint [8] and Spirent [25]. These solutions typically fulfill purposes such as testing loading-balancing, routing, or firewall systems. In these settings, low emphasis is placed on achieving behavior "realism" at the host layer in the dataset – it matters less that these traffic streams were not generated by human operators.

2.3 Anonymization paradigms

Existing network trace anonymization techniques typically fall into one of two paradigms: query-based systems which control the data on a protected server and allow users to make restricted queries, and systems based on one-way transforms that modify the original data and release an anonymized version. The former model has been studied more frequently in the domain of statistical databases, such as medical and health records, while the latter standard is more applicable to dissemination of network traffic traces

[1]Mathematical kernel: inner-product in a Hilbert-space

where the value of the data is not so easily broken down and quantified into sufficient statistics (such as means and deviations.) Often, especially in the security context, the utility of the data is unknown until researcher had a chance to examine it. Therefore, in these settings, raw, minimally-tampered data is often desired. This section explores previous work in network-data anonymization, discuss the importance of policy-driven systems, and concludes with some discussion on relevant attacks against modern anonymization systems.

2.3.1 Query-based systems

Query-based systems have been well researched in the past, and has observed the most progress in theoretical developments among the different approaches. Their success owes to the underlying assumptions and restrictions that makes up the foundation for this approach; specifically, that the value of the data can be broken down, and re-represented to the outside world as a closed set of quantifiers that users may makes queries against. For example, one may ask for the count, mean, or standard deviation for a particular entry in set of records. More quantifiers allow greater flexibility in processing. Though, regardless of how expansive this range of queries may be, this paradigm remains dependent on human operators to identify and construct these quantifiers *a priori*. More subtle is the assumption that the operator fully understands the full value and scope of the data a priori. The secure query-based approach proposed by Mirkovic is representative of this paradigm [18], where an SQL-like query language is presented to the user, and security policy enforcement is maintained on the server. Parate *et al.* [23, 22] advocates for an exchange of meta-level constraints and options between the provider and user where the user sets the constraints for utility such as port-consistency, and the server offers transformation policies such as encryption, re-ordering, *etc*, and reconciliation is handled server-side. The benefit of having a restricted system such as this is that metrics for diversity, entropy, and related measurements are often simple to derive and as a result, theoretical guarantees on anonymization are achievable. Beyond the difficulty of quantifying data into elements of utility *a priori*, the other downside to this approach is that centralized server side solutions are not scalable to large problem sets, where network traffic traces for just one enterprise-level environment may span terabytes [20].

2.3.2 Anonymization as a one-way transform

The second approach, which is far more popular for network data, transforms the data and releases the end-result. This process ideally cleanses the dataset of discriminative identifiers which might pierce privacy or anonymity. Many notable works in this area are worth mentioning. These are listed as here in the chronological order of their release. `Tcpdpriv` [17] is one of the earliest traffic-anonymization tools; simple packet-header sanitization tool, it was invented before modern de-anonymization attacks were known. `Tcpurify` [7] is another early, somewhat more advanced, sanitization tool developed for university use. The `tcpmkpub` [21] tool presented the first policy-driven anonymization framework offering a range of anonymization primitives, transparent handling of the tcp-stack, and advanced parsing functionality, all driven by a customizable anonymization language. `CryptoPan` [11] is the first provable prefix-preserving transformation method for IP addresses. `SCRUB-tcpdump` [31] allows one to extract only the useful properties of the dataset from trace. Traffic Morphing techniques presented by Wright *et al.* [28] attempts to mask the statistical signature of network protocols through data-padding to make them difficult to identify. Finally, the most recent advance in network trace anonymization is the PktAnon framework [12] by Gamer *et al.*, which, much like `tcpmkpub` is a policy-driven approach that allows users to customize a set of anonymization primitives for use on the data, using an XML-driven anonymization-policy language. Anonymization-policy driven frameworks is emerging as the most effective and promising of all anonymization methodologies for offline data anonymization. This following subsection explores this are in more detail.

2.3.3 Policy-driven frameworks

Policy-driven frameworks emerged from the realization that security, privacy, and utility for network trace data is a fluid continuum that is hard to define, and is not the same for every party. Further, anonymization and de-anonymization methods all operate using known- and unknown-information. There is no way to know for example, if the a particular flag setting in a particular packet uniquely identifies an individual, nor is it known if the removal of all flags is a reasonable anonymization operation. It is this realization that has led to the design of policy-driven anonymization frameworks which aim to abstract the lower-level packet-processing mechanics into a policy-driven language that users may customize to their needs. A set of anonymization transforms is provided to the user along with an intuitive language to drive these transformations, with best-use suggestions provided. It is further ideal that the data provider and the user should have some form of a contract that indicates what the data will be used for. This contract should guide the design of the anonymization policy [21, 6, 12]. Our research is grounded on this foundation: of privacy and anonymity in the presence of information uncertainty.

2.3.4 A balance between privacy and utility

A study of the state-of-the-art in network trace anonymization reveals that, ultimately, high quality research datasets should begin with live-captured traffic and, through a process of anonymization, private information should be removed while everything else should remain. The optimal anonymization procedure is simply to remove all information from the data; conversely, the optimal utility-preserving transformation makes no changes to the dataset. These are both undesirable extremes that sacrifice one goal for the other. One would expect that if utility does not require breach-of-privacy then the two should not be mutually exclusive. Further, privacy and utility are not universally defined; rather, they are often subjective. Privacy therefore, much like the rest of security field, revolves to a certain extent on risk-management [21], and policies which drive any anonymization procedure should result from a mutual understanding between the data provider and user. The ideal anonymization tool should satisfy all party's definitions of privacy, with minimal dilution of the fidelity of the dataset; providing the desired information, and nothing else.

3. NETWORK TRAFFIC MODELS

Network traffic modeling begins with feature extraction, where raw packet information from PCAP data is transformed into sets of numerical values that distinctively represent individual identities. We delineate traffic behavior into the following levels of granularity: **Protocol**: Data-transfer protocols, such as HTTP or SSH. Encompasses features such as size, frequency, and timing of packet exchanges. **Application**: A collection of network protocol usage, this layer encompasses type and frequency of protocols used. Web-browsers will typically use HTTP, and a typical connection sees a small request followed by a large response. **User/Host**: A collection of applications, and the characteristics of their usage, such as time of use, frequency, and duration. Distinctive user-behavior emerges at this level, as well as daemon services such as mail servers. **Network**: A collection of host behaviors uniquely defines

the profile user-interaction on a given network.

Various works exist in classification tasks at different layers. This includes Wright *et al.* [29] in demonstrating that protocol-level behavior is distinct, and that HTTP, FTP, *etc* can be uniquely identified even when the contents of these channels are encrypted. This and related works typically model the size, timing, and frequency of packet exchanges as statistical signatures.

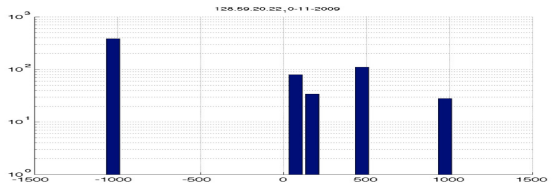
The scope of this study is anonymization at the *host* layer, and we seek to capture statistical representations of behavior for users and network servers (referred to, colloquially, as "hosts" in this paper.) True labelings for interaction between these layers do not exist. For example, we may guess that user A is using a web-browser if we observe outbound requests to port 80 on host B. Such common signatures, however, do not exceed but a handful; we further might not know if hosts A and B are users, or service daemons, or both. What network researchers typically do have, in the best scenario, is access to packet-level data such as PCAP dumps captured by a tool like `tcpdump`. Working with packet-level data gives us a bottom-up view of network traffic characteristics, and the true interactions between the network, user, application and protocol-layers are hidden from us. What is needed is a cross-layer model that can accurately classify hosts level behavior using raw data at the level of packets.

3.1 Statistical representations of "behavior"

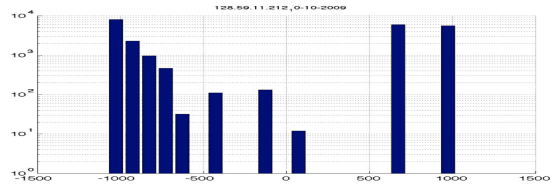
This paper proposes to abstract the interactions described above using state-based models, with the unknown-interactions modeled using hidden states. This section describes the foundation for our work: we propose a representation of host behavior using meta-content, namely packet headers and Netflow, to construct port-histogram profiles, and extend these into the real-time domain using times-series models. Among the models we propose are our variations on the hidden Markov model and our Protocol-Transition model. Meta-data such as packet headers is useful for several reasons: primarily, content is usually not provided with the dataset and packet headers are all that is available. Secondly, we aim to develop methods that characterize behavior which, at higher levels, is independent of content – a user's web browsing behavior (time, duration, frequency, *etc* may be idiosyncratic, even if the pages he/she view differs each day.)

Our main results demonstrated that host-layer behavior can be uniquely represented by volumetric and transitional statistical features for sent-to and received-on ports on a host, paired with time-series models. The patterning and its time-series interpretation distinguishes one host from another, and given sufficient sampling distinct *statistical fingerprints* of behavior emerge.

Figure (1) shows an example of this pattern. 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. However, a holistic view of traffic at the granularity of a single day could be too coarse to truly capture the intricacies of behavior shifts. What is needed is a time-varying view of the host, and how these activities change across the span of hours, days, weeks. A natural extension of the port-histogram model is to capture the histogram on smaller intervals and use a collection of such intervals as the feature set. This leads to the



(a) User machine port-histogram



(b) Server port-histogram

Figure 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.

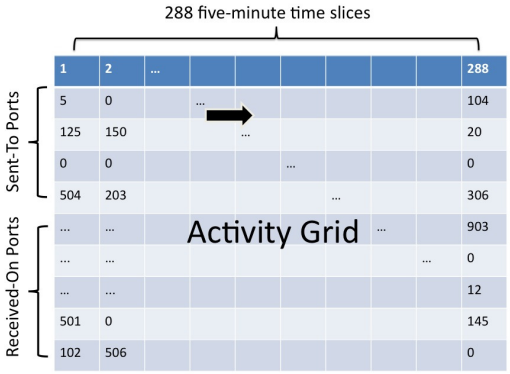


Figure 2: The Activity-Grid: 288 five-minute port histograms; a time-series feature representation for network traffic.

construction of the time-series histogram data structure. Figure (2) shows what we call an "Activity grid", a dataset structure composed of 288 distinct port histograms, captured at five-minute intervals throughout the day; beginning at 00:00:00 of the day and ending at 23:59:59 (Hour:Minute:Second). Using 2050 features to represent input and output port-activity gives us a 2050×288 matrix.

3.1.1 Sparsity, and dimensionality reduction

For the vast majority of hosts, the 2050×288 activity matrix is sparse, with 90% of the entries unpopulated on average. This is due to the host not observing any traffic in certain ports, and/or at certain times. This sparsity yields two sets of problems. First, underfitting leads to poor generalization performance. Secondly. memory requirements increase – if we used one full matrix per training day, roughly 2Mb worth of memory would be required per host. A trace file for enterprise-level traffic dumps typically consists of thousands of hosts, and in our tests using Columbia's traffic with roughly 7,000 hosts, full-memory consumption and disk-thrashing occurs roughly 10 seconds into processing under this naive setting. To solve these problems, we use dimensionality-reduction algorithms to project the data into a smaller subspace. The direct approach is to use subspace projection-based algorithms. This con-

sists of learning a projection matrix $\mathbf{P}$ such that the new samples are derived by the following linear relation: $\mathbf{Y} = \mathbf{P}^\top (\mathbf{X} - \mu), \quad \boldsymbol{\mu} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{X}_i$. Where $\mathbf{X}$ is an instance of an activity matrix. As this is a linear projection, the columns of $\mathbf{P}$ are constrained to be orthonormal. Using Principal Component Analysis (PCA), the columns of $\mathbf{P}$ consists of the top eigenvectors of the covariance matrix $\boldsymbol{\Sigma}$, recovered from the samples: $\boldsymbol{\Sigma} = \sum_{i=1}^{N} (\mathbf{X}_i - \boldsymbol{\mu})(\mathbf{X}_i - \boldsymbol{\mu})^\top$. Additionally, the Nyström method for low-rank approximations can be used to obtain an estimate for the covariance matrix at significantly less computational costs, with some sacrifice of precision. When computing $\boldsymbol{\Sigma}$ is still prohibitively expensive, Random-Projection (RP) may be used, where $\mathbf{P}$ is randomly initialized and an orthonormality step is applied using the *Gram-Schmidt* process. For certain distributions, the information-loss in not prohibitive and can be bounded (see Dasgupta [10]). A more effective kernel-based non-linear extension is discussed in a subsequent section.

3.2 Exponential-family behavior models

In this paper, we study *discrete time-series models with Markov transitions.* This scope is appropriate for two reasons. First, we consider network behavior to be Markovian in nature, in that the current state of the system is determined by its previous state. This property is self-evident given the nature of network traffic, in that events are not isolated, especially not at the packet-level. When an event occurs (web-site downloaded, mail sent, *etc*) an interaction of packet-traffic occurs, at the TCP layer with a SYN/ACK hand-shake to establish the session, or an HTTP-request at the application layer. Secondly, given that behavior is not static, a time-varying model is more appropriate to capture behavior-transition.

This type of model can be interpreted as statistical finite-state machines where, within each state, there is a distribution for a particular output symbol that can be a discrete value from an alphabet or a real-valued output. As the system progresses through the states, guided by a separate transition-probability model, a string of outputs known as the "observation sequence" is produced. In the case of hidden-state machines, such as the hidden Markov model (HMM)[24], the state of the machine at any given time is *unknown*. Inference is performed using efficient algorithms (such as the Junction Tree Algorithm) to calculate the likelihood given all possible state-transition configurations. The complexity of a time-series model is determined by the number of (hidden) states, the *transitions* model, and the *emissions* model at each state. The emissions model defines the statistical distribution from which data at a particular time step ($\mathbf{x}_t$) is implicitly drawn from. For example, if we model human movement, there may be emissions models for discrete classes of posture when walking (upright, bent, arms forward, leg-back, *etc*); the transitions model defines how these emissions flow between each other. Learning a time-series model consists of finding the optimal configuration for all of the parameters of the time-series model given a corpus of training data (observations). A transition table is typically used to model the state-transition probabilities for these types of models. This row-normalized (sums to 1) table tracks the probability of transitioning from one state to another using a single scalar value $\in [0, 1]$.

The question then becomes, what emissions model is appropriate for network behavior. For reasons which will become self-evident, our work focuses on the exponential-family of models: particularly the Multi-variate Gaussian, Multinomial, and Discrete Probability Table models. In addition to favorable empirical performance, this class of models can be optimized in the same way, using maximum-likelihood or expectation-maximization (EM) methods. A study of the performance of these models provides insights into network traffic modeling and drives the derivation of an improved model, described in a later sub-section.

When we characterize behavior as a ratio of services executed, the Multinomial model is most appropriate:

$$p(\mathbf{x}|\theta) = \frac{n!}{\mathbf{x}(1)!, ..., \mathbf{x}(k)!}\theta(1)^{\mathbf{x}(1)} \cdots \theta(k)^{\mathbf{x}(k)}. \tag{1}$$

Here, $\sum_{i=1}^{k} \mathbf{x}(i) = n$, and the parameter variable θ constitutes the normalized ratios of events $\mathbf{x}(i), ..., \mathbf{x}(k)$ in the training data. The Multinomial distribution models discrete event occurrences as separate independent events, and tracks the proportion of occurrence to one-another. This entails fingerprinting a host's network behavior based on the ratio of services invoked, at each time step. When it's possible to label a state of a system using symbols from a finite alphabet {"work","check email","web surf",...}, through human-intervention or an automated labeling process, Discrete Conditional Probability Table (DCPT) model is more appropriate. This approach models the likelihood of a sequence of symbols, and the transitional probabilities between one symbol to another:

$$p(a|b) = \mathbf{T}(a, b), \qquad \sum_{j} \mathbf{T}(i, j) = 1 \quad \forall i. \tag{2}$$

Maximizing the joint probability of symbols given their order of appearance in the training data optimizes the parameters of this model. Finally, when little information is known about the data, a classic choice is the Multivariate-Gaussian distribution, also known as the Multivariate-Normal (MVN) distribution. This model considers the emission as a single independent-identically-distributed (i.i.d.) multivariate variable. Due to the Central Limit Theorem, a Multivariate-Gaussian assumption is most appropriate when little domain-knowledge is available. The MVN is also a continuous model, which allows flexibility in working with feature transformations and other preprocessing functions to optimize results. Let $\theta = \{\mu, \Sigma\}$, we have:

$$p(\mathbf{x}|\theta) = \frac{1}{(2\pi)^{d/2}\sqrt{|\Sigma|}} \exp\left\{-\frac{1}{2}(\mathbf{x} - \mu)^\top \Sigma^{-1}(\mathbf{x} - \mu)\right\}. \tag{3}$$

3.3 Time-series extension using discrete hidden Markov models

Given the emissions models described in the previous section, we can perform the time-series extension by incorporating them into hidden Markov models. An HMM is a Markov model where the process transitions between states in an unobservable path. Consider a first-order stationary HMM with Gaussian emissions, the probability of a sequence is given as $p(\boldsymbol{X}|\theta)$, where $\boldsymbol{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_T\}$ is an observation sequence of length T, and each observation vector is $\mathbf{x}_t \in \Re^d$. This model assumes that $\mathbf{x}_t$ at each time step t was emitted from one of M independent emissions models ($q_t = \{1, \ldots, M\}$) which represent the state of the system at a particular time-step. The exact value of q_t is unknown and is referred to as a "hidden'" state. The transition through a sequence of hidden states $\boldsymbol{q} = \{q_1, \ldots, q_T\}$ generates the observed sequence $\boldsymbol{X}$. To obtain a normalized distribution, we sum over all possible values of $\boldsymbol{q}$:

$$p(\mathbf{X}|\theta) = \sum_{q_0, \ldots, q_T} p(\mathbf{x}_0|q_0)p(q_0) \prod_{t=1}^{T} p(\mathbf{x}_t|q_t)p(q_t|q_{t-1}). \tag{4}$$

This HMM is specified by the parameters: $\theta = \{\boldsymbol{\pi}, \boldsymbol{\alpha}, \boldsymbol{\mu}, \boldsymbol{\Sigma}\}$. These are the initial-state probability distribution $\pi_i = p(q_0 = i)$, $i = 1 \ldots M$, the state transition probability distribution denoted by a matrix $\boldsymbol{\alpha} \in \Re^{M \times M}$ where $\boldsymbol{\alpha}_{ij} = p(q_t = j|q_{t-1} = i)$, the emission density $p(x_t|q_t = i) = \mathcal{N}(\mathbf{x}_t|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$, for $i = 1 \ldots M$,

where $\boldsymbol{\mu}_i \in \Re^d$ and $\boldsymbol{\Sigma}_i \in \Re^{d \times d}$ are the mean and covariance of the Gaussian in state i. We use $\boldsymbol{\mu} = \{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_M\}$ and $\boldsymbol{\Sigma} = \{\boldsymbol{\Sigma}_1, \ldots, \boldsymbol{\Sigma}_M\}$ for short. d is the dimensionality of $\mathbf{x}$.

Brute-force evaluation of equation (4) is requires exponential time, however, dynamic programming methods such as the Junction Tree Algorithm (JTA) or the Forward-backward algorithm, can compute this quantity efficiently. Estimating the parameter (θ) of an HMM is then typically done via EM with random initialization. We present our derivations for the EM-update rules for the MVN-HMM here. The E-step uses a forward pass to obtain posterior marginals over the hidden states given the observations:

$$\gamma_t(i) = p(q_t = s_i | \mathbf{x}_n, \theta) \quad (5)$$

$$\xi_t(i,j) = p(q_t = s_i, q_{t+1} = s_j | \mathbf{x}_n, \theta). \quad (6)$$

A hat-variable ($\hat{\theta}$) represents a new estimation of that variable based on the previous iteration of EM. The M-step updates the parameters θ using these E-step marginals as follows:

$$\hat{\pi}_i = \gamma_1(i) \quad (7)$$

$$\hat{\alpha}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \sum_{j=1}^{M} \xi_t(i,j)} \quad (8)$$

$$\hat{\boldsymbol{\mu}}_i = \frac{\sum_{t=1}^{T} \gamma_t(i) \mathbf{x}_t}{\sum_{t=1}^{T} \gamma_t(i)} \quad (9)$$

$$\hat{\boldsymbol{\Sigma}}_i = \frac{\sum_{t=1}^{T} \gamma_t(i) (\mathbf{x}_t - \boldsymbol{\mu}_i)(\mathbf{x}_t - \boldsymbol{\mu}_i)^\top}{\sum_{t=1}^{T} \gamma_t(i)}. \quad (10)$$

The E and M steps are repeated until the likelihood value $p(\mathbf{X}|\theta)$ of the training dataset $\mathbf{X}$ with model θ does not improve. The above derivation yields the parameter update rules for the HMM with MVN emissions. As previously stated, the benefit of using distributions from the same exponential-family class means that deriving HMMs with Multinomial, DCPT, emissions are derived in the same way. These derivations are simple and (given that the MVN model performed best) are omitted for brevity.

3.3.1 Accuracy evaluation

This section presents empirical evaluations for our proposed models. We collected three weeks of worth of packet-header-only traffic covering 80 Columbia machines, chosen to represent a range of different network hosts, from database servers to web servers, to user machines. Since we do not have true labels for how hosts should be clustered together, we test this key component of the algorithm in a classification experiment to see if we can classify a host's own traffic correctly. Further on in this paper, we discuss clustering. As our clustering algorithm is strongly determined by the model parameters, classification performance is strongly correlated with clustering performance. And as we have no direct way to evaluate clustering results on unlabeled data, this classification set-up is most appropriate. We set up a classification experiment, splitting each user's traffic into randomized training and testing sets. Each test sample is matched against the model of every host in a highest-likelihood value wins setup, giving us an 80 versus 80 classification task. The baseline performance for this experiment, using random guessing, would yield 1.25% accuracy. Raw indicates unprocessed raw packet-count features. PCA indicates normalized PCA-projected features, and "Session" indicates PCA project features which were pre-processes to be gap-less – making no attempt to model off-periods in network activity. Sandia A and B represent similar experiments conducted in 2008 while Author 1 worked at Sandia National Laboratories. As Table (1) shows, we perform considerably better than random guessing, achieving a nearly 6,600% improvement over the baseline using the Gaussian model using vectors. In our experiments, 80% of the packet traffic for a host is randomly chosen as the training set, and 20% as the test set. The reported results are each averages of *five* independent runs.

	G-HMM	M-HMM	D-HMM	G-VEC.
CU RAW	—	22%	25%	82%
CU PCA	10%	—	—	63%
CU SESSION	28%	22%	25%	80%
SANDIA A	67%			
SANDIA B	91%			

Table 1: Performances of HMM models on CU's 80 vs. 80 classification task. Baseline (random guess) would give 1.25% accuracy. Dashed entries indicate infeasible settings. G-Vec. refers to a Gaussian vector model.

One important hypothesis emerges from these results: the sparsity of the data strongly influences performance; notice that as we remove the sparsity from the CU data the performance improved. Sandia's experiments were conducted on servers with very dense traffic output, compared to CU machines. Sandia A consists of intra-class classification (*e.g.* database vs database), Sandia B consists of inter-class (*e.g.* webserver vs database). Further, notice that the Gaussian vector model had the strongest performance, in this case the data is collapsed into a single vector emission (no transitions are used). This indicates that for low-density traffic datasets, a flat high-level model for behavior could suffer from under-fitting. The following subsection describes a new model designed specifically for datasets where this is an issue.

3.4 Protocol-Transition Markov Model

Instead of modeling the dynamics of network traffic using hidden-state models, a good alternative is to treat the system as a transition between specific application layer protocols (FTP → HTTP → SMTP, ...) With this approach the protocols identify the states explicitly, thus we no longer need to use hidden-state representations. Instead, we simply model the characteristics of the traffic observed at any given state, while simultaneously tracking state-transitions. To model the emissions behavior of these applications the same exponential family may be used, however given that MVN yields the strongest performance in the previous experiments, we chose the scalar-Gaussian mixture in this model:

$$p(x|\theta) = \sum_{i=1}^{M} \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left\{ -\frac{(x-\mu_i)^2}{2\sigma_i^2} \right\}. \quad (11)$$

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 EM. The update rules are derived in the same manner as the multi-variate Gaussian maximization step, except now the variance is scalar:

$$\hat{\sigma}_i = \frac{\sum_{t=1}^{T} \gamma_t(i)(x_t - \mu_i)(x_t - \mu_i)}{\sum_{t=1}^{T} \gamma_t(i)}. \quad (12)$$

The E-Step for computing the γ, ξ marginals remain the same. Since the states are not hidden in this case, parameter estimation for the state-transition parameter $p(s_t|s_{t-1})$ is much simpler and consists

of simply tracking the transition-frequencies as follows:

$$p(s_b|s_a) = \frac{\#(s_i = b, s_{i-1} = a)}{\#(s_i = b, s_{i-1} = a) + \#(s_i \neq b, s_{i-1} = a)}. \quad (13)$$

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. Putting these together we obtain the probability distribution for the protocol-transition model:

$$p(\mathbf{s}, \mathbf{x}|\theta) = p(s_1)p(\mathbf{x}(1)|s_1)\prod_{t=2}^{T} p(s_t|s_{t-1})p(\mathbf{x}(t)|s_t). \quad (14)$$

To bridge the gap between this model and network data, a few additional modifications are required. First and foremost, since PCAP data does not come in the form of traffic-volume/protocol statistics, the individual packets must be reassembled into sessions in order for the needed statistical information to be derived. An easy solution for this is to use a pcap-to-Netflow conversion tool. In our experiments `softflowd` [16] was used to perform this conversion. From the Netflow format, we can extract the features needed to train the PT model. Secondly, given that there are 64k available ports in TCP/IP, we could, in theory, have a $65,536 \times 65,536$-sized transition matrix, which would be impossible to estimate accurately. To solve this problem, we assume that all ports above 1024 to be ephemeral – baring no correlation with application-layer traffic, and therefore can be collapsed into a single entry in the transition table (more complex delineations are possible given domain knowledge.) Thirdly, if 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.

3.4.1 Accuracy evaluation

A range of datasets were used for this experiment, these include the well-known LBNL dataset [3], collections from West Point and NSA [2], and University traffic from CU and GMU. For each host, multiple 150-entry segments of Netflow were sampled for testing data, while the rest was used for training. The model that yields the highest likelihood on the test sample labels that sample. If there are 100 hosts, a baseline would yield a 1% chance of a correct random guess. Table (2) shows the performance of the PT-Markov model. This model consistently achieves significantly higher performance than the baseline, re-enforcing our hypothesis that deriving a lower-level model at a finer granularity, where sparsity is less evident, improves performance. The use of Netflow-like statistics permits training with significantly less data than HMM-based approaches, which could require data on the order of days. This lower level model requires sessions on the order of minutes to hours, and works very well even in cases of very sparse training data. This allowed us to evaluate two additional CU machines which had very sparse traffic that could not be evaluated in previous experiments. Our results show that given any traffic sample, on the order of 20 minutes to an hour, our models can attribute that traffic to the true host, from among 82, with 76% accuracy. 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 PCAP to Netflow-like statistics can lead to inaccurate records on session duration, size, *etc.*, and this inaccuracy influenced poorer performance on our model.

4. NETWORK TRACE CLUSTERING AND SIMILARITY VISUALIZATION

4.1 Time-series kernels and Kernel-PCA

A very efficient general kernel between hidden Markov models is the probability product kernel (PPK). First introduced by Jebara *et. al* [13], the generalized inner product of this kernel is found by integrating a product of two distributions over the space of all potentially observable data sequences $\mathcal{X}$:

$$\mathcal{K}(p(\mathbf{x}|\theta), p(\mathbf{x}|\theta')) = \int_{\mathbf{x}} p^{\beta}(\mathbf{x}|\theta)p^{\beta}(\mathbf{x}|\theta')d\mathbf{x}. \quad (15)$$

When $\beta = 1/2$, the PPK becomes the classic *Bhattacharyya affinity* metric between two probability distributions. This kernel is different than previously proposed kernels, such as the Mutual-Information/Cross-likelihood kernel by Yin *et al.* [30], in that the inner product is solved as a direct function of the parameters of the model, as opposed to traditional kernels which use the data sequences. This translates to a kernel that is both faster to compute, and have comparatively much lower memory requirements than the original. For certain distributions, the PPK can be computed analytically instead of numerically, greatly reducing computational costs. When the probability distribution $p(\mathbf{x}|\theta)$ of the integral in Eq. (15) is a time-series model, such as an HMM, two elements of integration are required. For brevity, we denote $p(\mathbf{x}|\theta)$ as p and $p'(\mathbf{x}|\theta')$ as p'. The first component is the evaluation of what is known as the *elementary kernel* $\Psi(\cdot)$, which is the Bhattacharyya affinity between emissions models for p and p' integrated over the space of all emissions $\mathbf{x}$:

$$\Psi(\theta, \theta') = \int_{\mathbf{x}} p^{1/2}(\mathbf{x}|\theta)p^{1/2}(\mathbf{x}|\theta')d\mathbf{x}. \quad (16)$$

For the exponential family of distributions, such as those described in § 3, these integrals can be solved analytically. In the case of full time-series models with Gaussian emission-models, $\Psi(\theta, \theta')$ is solved as follows:

$$\mu^{\dagger} = \Sigma_i^{-1}\mu_i + {\Sigma_j}^{-1}\mu_j \quad (17)$$

$$\Sigma^{\dagger} = (\Sigma_i^{-1} + {\Sigma_j}^{-1})^{-1} \quad (18)$$

$$\mathbf{Z} = \mu_i^{\top}\Sigma_i^{-1}\mu_i + {\mu_j}^{\top}{\Sigma_j}^{-1}\mu_j - {\mu^{\dagger}}^{\top}\Sigma^{\dagger}\mu^{\dagger} \quad (19)$$

$$\Psi(i,j) = \frac{|\Sigma^{\dagger}|^{1/2}}{|\Sigma_i|^{1/4}|\Sigma_j|^{1/4}\exp\left(-\frac{1}{4}\mathbf{Z}\right)}. \quad (20)$$

In the case of HMMs, the PPK of Eq. (15) requires integration over their hidden-states in addition to their emissions models. An efficient method for doing this is to take advantage of the interaction between these hidden-states to factor the equation in a way that admits an iterative solution. Such a solution is provided in Table (3).

Given the elementary kernel, the PPK between two HMMs is solved in $\mathcal{O}(TM^2)$ operations using the formula in Table (3), where T is the length of the sequence and M is the number of hidden states. Further details about the PPK can be found in [13, 14]). Given a kernel affinity between two HMMs, a non-parametric relationship between time series sequences emerges. It is now straight forward to apply non-parametric clustering and embedding methods which will be described in in the following subsection.

The elementary kernel for the Protocol-transition model is similarly derived. The main difference is the addition of a second layer of interacting variables since each model represents a *mixture* of scalar Gaussian distributions. The integral is solved by expanding the product and following the same complete-the-squares technique as described in [13]. Due to the linearity of integration,

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 2: Accuracy evaluations across different datasets.

Probability product kernel $\mathcal{K}(p, p')$:

Elementary kernel $\Psi(\theta, \theta') = \int_{\mathbf{x}} p^{1/2}(\mathbf{x}|\theta) p^{1/2}(\mathbf{x}|\theta') d\mathbf{x}$

$\Phi(q_0, q'_0) = p(q_0)^{1/2} p'(q'_0)^{1/2}$

for $t = 1 \dots T$

$\quad \Phi(q_t, q'_t) = \sum_{q_{t-1}} \sum_{q'_{t-1}} p(q_t|q_{t-1})^{1/2} p(q'_t|q'_{t-1})^{1/2} \Psi(q_{t-1}, q'_{t-1}) \Phi(q_{t-1}, q'_{t-1})$

end

$\mathcal{K}(\theta, \theta') = \sum_{q_T} \sum_{q'_T} \Phi(q_T, q'_T) \Psi(q_T, q'_T)$

Table 3: FAST ITERATIVE ALGORITHM FOR THE PROBABILITY PRODUCT KERNEL

the elementary-kernel of a linear-mixture distribution, such as the Protocol-Transition model, can be solved as a function of pair-wise sub-elementary-kernels $\widehat{\Psi}$, on the sub-models of the mixture. The sub-elementary-kernel for a scalar-Gaussian is derived here. Let $\mu_{i,m}$ and $\sigma_{i,m}$ represent the m^{th} sub-model of the i^{th} Gaussian mixture, then we have the following:

$$\mu^{\dagger} = \mu_m/\sigma_m + \mu_n/\sigma_n \quad (21)$$

$$\sigma^{\dagger} = (\sigma_m^{-1} + {\sigma_n}^{-1})^{-1} \quad (22)$$

$$\mathbf{Z} = \mu_m \sigma_m^{-1} \mu_m + \mu_n {\sigma_n}^{-1} \mu_n - \mu^{\dagger} \sigma^{\dagger} \mu^{\dagger} \quad (23)$$

$$\widehat{\Psi}(\theta_m, \theta_n) = \frac{\sqrt{\sigma^{\dagger}}}{(\sigma_m \sigma_n)^{1/4} \exp\left(-\frac{1}{4}\mathbf{Z}\right)} \quad (24)$$

$$\Psi(i, j) = \sum_m \sum_n \alpha_m \gamma_n \widehat{\Psi}(\theta_{i,m}, \theta_{j,n}). \quad (25)$$

α, γ represent the mixture coefficients of the i, j mixtures, respectively. Note that the number of sub-models need not be the same between two PT models, as long as each model is properly normalized (*i.e.* $\sum_i \alpha_i = 1$). Given the elementary kernel, computing the full PPK for the PT model is similarly completed by using the algorithm given in Table (3).

Note that our kernel-based framework allows one to use *any* proper Mercer kernel, not just PPK. For example, a simpler cross-likelihood kernel is also available. Shown in Eq. (26), this kernel calculates the Hilbert-space inner-product as a function of the likelihood values from two densities (θ_p, θ_q), evaluated over a finite set of training data. This kernel takes the following form:

$$\mathcal{K}(p(\mathbf{x}|\theta_p), p(\mathbf{y}|\theta_q)) = \exp\left\{-\frac{||p(\mathbf{y}|\theta_p) - p(\mathbf{x}|\theta_q)||}{2\sigma^2 p(\mathbf{x}|\theta_p) p(\mathbf{y}|\theta_q)}\right\}. \quad (26)$$

The main benefit of this kernel is the ease of implementation, requiring only that likelihood-evaluation is possible given a model. The downside of this kernel is the calculation overhead incurred in having to explicitly recover the likelihood values for each data sample used in the kernel estimate – for large sets of long data sequences, this cost can be prohibitively expensive. The PPK is proven to be orders of magnitude faster than this kernel [14].

4.2 Spectral Clustering for time-series data

Given any time-series model $p(\mathbf{x}|\theta)$, the most straight forward way to cluster the data is to extend the model using a linear mixture $\sum_i \alpha_i p(\mathbf{x}|\theta_i)$ and use a soft-max algorithm such as Expectation-Maximization to perform the clustering. However, for time-series models, this type of clustering is prohibitively expensive and it requires exponentially growing number of evaluations of the dataset. Further, by using a linear-mixture, one adopts an implicit assumption that the overall distribution of data takes on a parametric form.

Recent innovations in clustering has emerged, based on graph theory. And Eigen-based solutions to such problems has led to new classes of Spectral Clustering (SC) algorithms. In our context, an {HMM, PT} model is trained for each data-sequence, the PPK is then evaluated between all pairs of models to generate an Affinity matrix, which represents a graph. A spectral graph-cut procedure (shown below) segments this graph into distinct subgraphs (clusters). This leverages both parametric and non-parametric techniques in the clustering process, by making use of parametric models (HMMs, *etc*) to represent the data, yet makes no assumptions about the *overall* distribution of these models, allowing very non-linear clusterings. A more detailed explanation of SC is omitted, due to space constraints; the reader is referred to Ng & Weiss [19] for additional reference. Listed below are the steps of our SC-PPK algorithm, which is a time-series analogue of the Ng & Weiss SC algorithm, extended using time-series kernels.

The SC-PPK algorithm:

1. Fit a model {HMM,PT} to each of the $n = 1 \dots N$ time-series sequences to retrieve models $\theta_1 ... \theta_N$.
2. Calculate the Gram matrix $A \in \mathbb{R}^{N \times N}$ where $A_{m,n} = \mathcal{K}(\theta_m, \theta_n)$ for all pairs of models using the probability product kernel (default setting: $\beta = 1/2$, T=10).
3. Define $D \in \mathbb{R}^{N \times N}$ to be the diagonal matrix where $D_{m,m} = \sum_n A_{m,n}$ and construct the Laplacian: $L = D^{-1/2} A D^{-1/2}$
4. Find the K largest eigenvectors of L and form matrix $X \in \mathbb{R}^{N \times K}$ by stacking the eigenvectors in columns. Renormalize the rows of matrix X to have unit length.
5. Cluster the N rows of X into K clusters via k-means or any other algorithm that attempts to minimize distortion.
6. The cluster labels for the N rows are used to label the corresponding N time-series models.

Empirically, our algorithm greatly outperforms fully-parametric models such as mixtures-of-HMMs [14].

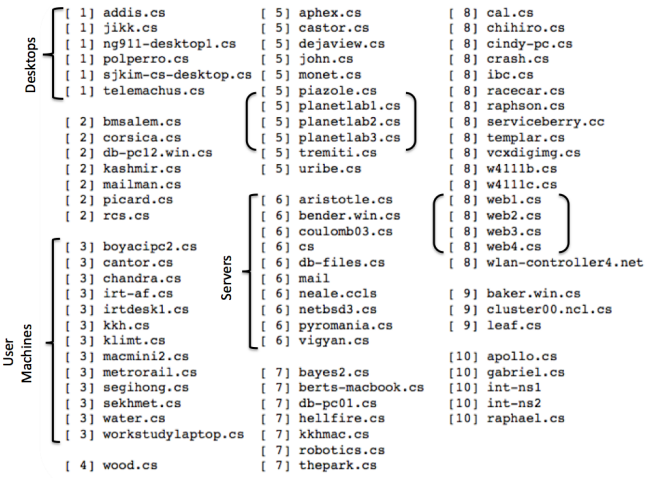


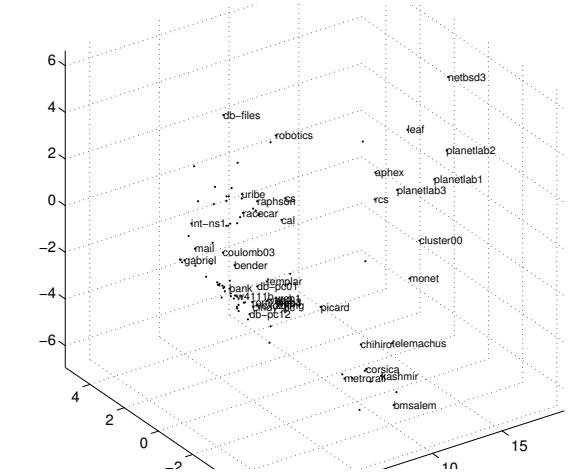
Figure 3: Sample HMM-clustering of host behaviors. Some obvious clusters are annotated.

Figure (3) shows the output of a clustering. While we do not have true labels for host behavior, we can infer some notion of accuracy by using a reverse DNS-lookup and checking their domain names. Here, we see groups of what appears to be similar hosts, such as the the web servers (web1, web2,...), as well as the Planetlab research machines, and user workstations (boyacipc2, irtdesk1). Based on our experience with Columbia's IT department, we recognize groups [9] and [10] as clusters of server machines. Overall, the initial results of this algorithm looks promising. A single outlier ([4] wood.cs) is generated, though this result is from an earlier incarnation of our algorithm, later versions no longer allow unbalanced clusters such as this.

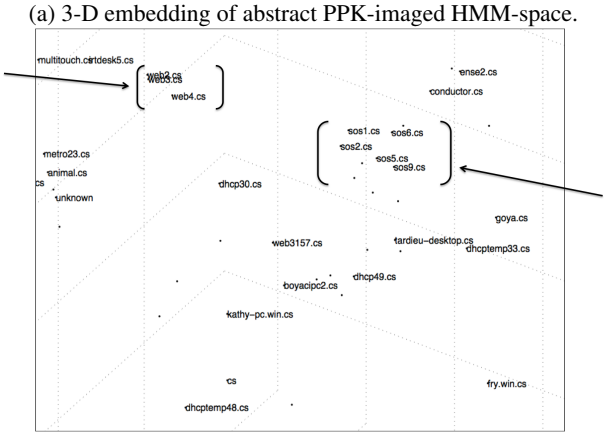
4.3 Network-behavior visualization with Low-dimensional embedding

All data: scalar, vector, time-series, or otherwise, exist as single points in some high (possibly infinite) dimensional Hilbert space. Low-Dimensional Embedding is a class of techniques that seek to capture a re-setting of these data points in a lower (typically 2-D or 3-D) dimension, while maintaining – to the best extent – the same pair-wise distances as the original data points. These techniques permit visualization of high dimensional data, and can also help confirm clustering patterns. Given a gram matrix $\mathbf{K}(i,j) := \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j)$ – the derivation of which was discussed in the previous sub-section – many embedding algorithms are immediately available. The most well known is Kernel Principal Component Analysis (KPCA). Given our derivations of the PPK for the HMM and PT models, KPCA for time-series network data is available for both data dimensionality reduction as well as visualization tasks. Dimensionality-reduction, in this context, is analogous to the previously mentioned PCA method, which was used to compress the Activity-grid input data. Whereas PCA finds a linear projection into a sub-space that maximizes the intra-class covariance, kernel-PCA finds an implicitly non-linear mapping by solving the same problem in the image of the non-linear mapping Φ induced by the kernel function:

$$\mathbf{C} := \frac{1}{m}\sum_{i=1}^{m} \Phi(x_i)\Phi(x_i)^\top.$$



(a) 3-D embedding of abstract PPK-imaged HMM-space.



(b) Clusters of similar hosts are immediately recognizable.

Figure 4: Kernel-PCA embedding of HMM-measured manifold. "planetlab", "web", and "sos" clusters are easily visible.

The eigenvectors of $\mathbf{K}$ spans the Φ-images of the training data, the solution set $\mathbf{v}$ takes the form $\mathbf{v} = \sum_{i=1}^{m} \alpha_i \Phi(x_i)$, which leads to a familiar eigenvalue decomposition problem:

$$m\lambda\boldsymbol{\alpha} = \mathbf{K}\boldsymbol{\alpha}, \qquad \mathbf{K}(i,j) := \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j).$$

The n^{th} component of the kernel PCA projected data point is then:

$$\langle \mathbf{v}(n), \Phi(\mathbf{x}) \rangle = \sum_{i=1}^{m} \boldsymbol{\alpha}_n(i)\mathcal{K}(\mathbf{x}_i, \mathbf{x}). \tag{27}$$

Where the kernel function $\mathcal{K}(\cdot,\cdot)$ can be either the probability product kernel or the cross-likelihood kernel. When projecting onto the top three principal components, for example, KPCA would provide a 3-D view of the data, as Fig. (4) shows. Other similar algorithms such as Multi-dimensional Scaling [15], Maximum Variance Unfolding [26], require only the $\mathbf{K}$ matrix as the primary input. Low-dimensional embedding allows us to view network traffic on a 2D/3D plane, to identify clusters of similar or anomalous behavior, as well as provide insights into how many clusters might exist in the dataset. A 4-dimensional view of this behavior manifold is also possible, by examining the embedding *across time*. With a time-lapse embedding we see how patterns of behavior can evolve, and how hosts approach and drift in behavior. A video demonstration of this is available on our website[2].

[2]http://www.cs.columbia.edu/∼yingbo/MVE-DHS-2.mov

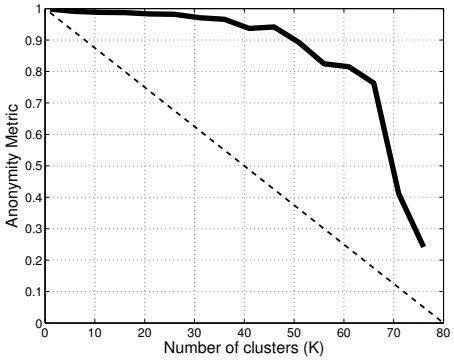


Figure 5: IMPACT: "Anonymity" vs Number-of-clusters.

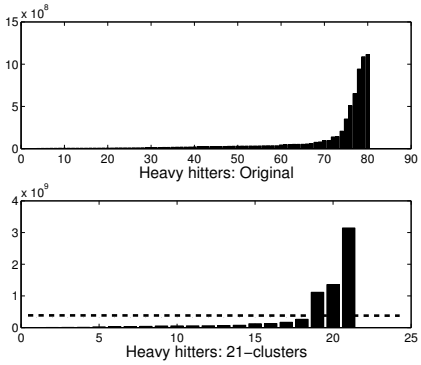


Figure 6: Heavy-Hitters: before and after. IMPACT preserves statistical distribution; dashed line shows random even-mix.

5. NETWORK TRACE ANONYMIZATION

5.1 The IMPACT framework

The Internet-Measured Private Anonymously Clustered Traces (**IMPACT**) framework is the realization of the results in this paper. IMPACT operates through the following steps: First, the traffic for all hosts in the network trace are extracted to produce statistical features, which is then used to train time-series models using the methods described in § 3. These models represent points embedded in an abstract-space (visualizable with techniques described in § 4.3. The spectral clustering algorithm described in § 4 clusters these models and provides a delineation of behavior classes for hosts within a network trace, providing confidence that merging members of this group would provide anonymity to the members while minimizing information-dilution within that class. PCAP-clustering is actuated in the `Merge` module of IMPACT which processes the network trace and reassigns each member of the cluster a single group IP address. All relevant parameters such as timestamps are adjusted to be consistent. Optional filtering and pre-processing modules are used to remove artifacts in the dataset. This de-noises the data of fragmented packets, Bogon packets, and other unneeded network oddities. For technology integration purposes, the first version of IMPACT was designed to be a module within `PktAnon`, allowing us to use their well-developed processing chains to further anonymize the data, using primitives such as the `CryptoPan` and other functions. Graphics which break down IMPACT's modules and functionality are provided in the Appendix. Fig. (9) shows the graphical user interface which we have designed for IMPACT and `PktAnon`. This technology allows users to select multiple trace files, customize the feature extraction and clustering algorithms, selecting the number of clusters for example, and adjust transformation primitives such as transformation settings for packet header contents in a policy-driven system.

Utility-preservation is, unfortunately, not simple to evaluate, given that "utility" and "anonymity" are largely subjective. However, we can achieve some measure of anonymity by measuring the entropy-gain given IMPACT's mixing transform. Appealing to the Anonymity-Set-Size estimate, we measure the ratio of one entity's packet-count to the resulting packet-count of the group-entity. Smaller ratios indicate more anonymity as the original becomes "lost in the crowd." An inverse *harmonic* mean of these ratios quantifies this concept. Fig. (5) shows results on the Columbia dataset of 80 hosts. We have the highest degree of anonymity when everyone is grouped into one cluster and the lowest when 80 clusters are used – as expected – but notice a favorable fall-off in the plot, which indicates that our system provides better groupings than purely random clustering, represented by the dashed line. Fig. (6) shows the statistic-preserving properties; in a study on measuring Heavy-Hitters on a network, we simply see a coarser view of the original density post-mixing; a purely random even-mix would yield the uninformative flat distribution represented by the dashed line.

5.2 Conclusion

Our approach to anonymization is founded on the hypothesis that statistical models for static and temporal traffic behavior may be used to identify groups of individuals with distinctively similar behavioral patterns. From there, we draw upon the concept of mix-nets for anonymization. In a mix-net, the traffic from multiple users are aggregated at smaller sets of exit points. `Tor` for example, is a well known anonymity provider that concentrates all traffic which enters the network through small set of exit nodes, using onion-routing within the network to provide end-to-end anonymity. The traffic aggregation in the Tor network does not, however, consider the behavior profiles of the users in the switching logic. It and similar solutions do not try preserve statistical features in the resulting traffic. By using machine learning algorithms to profile and cluster hosts based on behavioral characteristics, we apply a pseudo-mixed-net transform to *offline* data, while simultaneously preserving, in the aggregate, the statistics that characterize the members of individual groups. In this paper, we demonstrate quantifiable performance improvements over a purely randomized approach.

As part of our ongoing work, we can also achieve an interpolation effect on the profiles by partially-mixing traffic among different hosts. The kernel-based approach extends to kernel-regression and interpolation naturally, allowing us to "patch" missing traffic. We can transform user traffic to appear 30% similar to one profile and 70% to another, or some other desired ratio, by statistics-driven mixing of raw traffic in the desired ratios. These properties extends into a new field of statistics-driven traffic synthesis. Full papers on these topics are forthcoming.

Acknowledgments

We thank the following individuals for their contributions to this effort: Steve Bellovin, Dan Rubenstein, Vishal Misra, Tal Malkin, Eli Brosh, Kyung-Wook Hwang, Bert Huang, Blake Shaw, Krzysztof Choromanski, and Oladapo Attibe. We thank Prof. Angelos Stavrou and Brian Schulte at GMU for providing us with valuable data. 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).

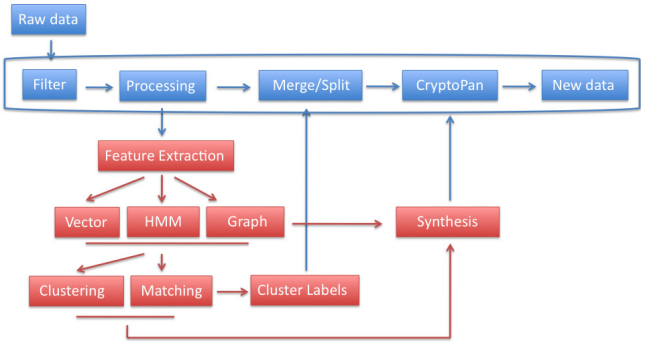


Figure 7: IMPACT modules.

Appendix

6. REFERENCES

[1] 1st ACM Workshop on Network Data Anonymization (NDA 2008), October 2008.

[2] ITOC CDX Research Dataset. http://www.icir.org/enterprise-tracing/, 2009.

[3] LBNL/ICSI Enterprise Tracing Project. http://www.icir.org/enterprise-tracing/, 2011 2011.

[4] Openpacket.org: a centralized repository of network traffic traces for researchers. https://www.openpacket.org, 2011.

[5] PREDICT: The Protected Repository for the Defense of Infrastructure Against Cyber Threats. http://www.predict.org, 2011.

[6] ALLLMAN, M., AND PAXSON, V. Issues and etiquette concerning use of shared measurement data. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement* (New York, NY, USA, 2007), IMC '07, ACM, pp. 135–140.

[7] BLANTON, E. TCPurify: A "Sanitary" Sniffer. http://masaka.cs.ohiou.edu/ eblanton/tcpurify/, January 2008.

[8] BREAKINGPOINT SYSTEMS. Breakingpoint. http://www.breakingpointsystems.com/, 2011.

[9] BRUGGER, T. KDD Cup '99 dataset (Network Intrusion) considered harmful. http://www.kdnuggets.com/news/2007/n18/4i.html, September 2007.

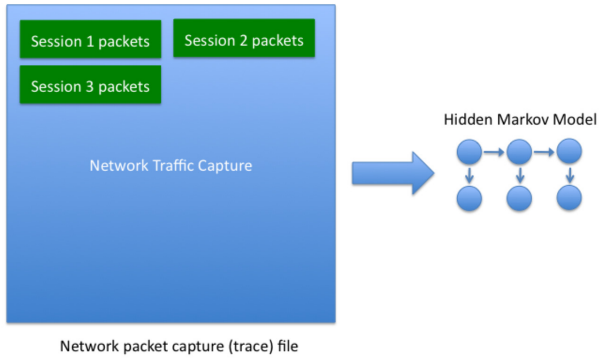
[10] DASGUPTA, S. Experiments with random projection. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence* (San Francisco, CA, USA, 2000), UAI '00, Morgan Kaufmann Publishers Inc., pp. 143–151.

[11] FAN, J., XU, J., AMMAR, M. H., AND MOON, S. B. Prefix-preserving ip address anonymization: measurement-based security evaluation and a new cryptography-based scheme. *The International Journal of Computer and Telecommunications Networking 46*, 2 (October 2004).

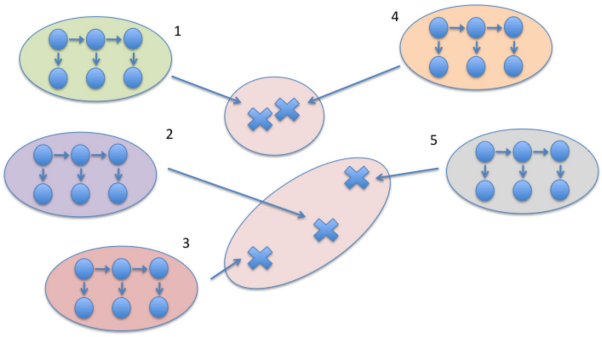
[12] GAMER, T., MAYER, C. P., AND SCHÖLLER, M. Pktanon - a generic framework for profile-based traffic anonymization. *PIK Praxis der Informationsverarbeitung und Kommunikation 2* (June 2008), 67–81.

[13] JEBARA, T., KONDOR, R., AND HOWARD, A. Probability product kernels. *Journal of Machine Learning Research 5* (2004), 819–844.

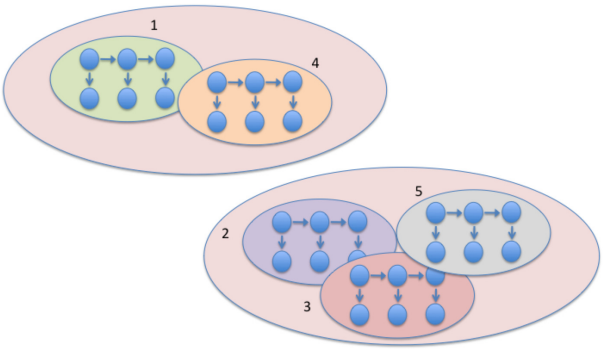
[14] JEBARA, T., SONG, Y., AND THADANI, K. Spectral clustering and embedding with hidden markov models. In



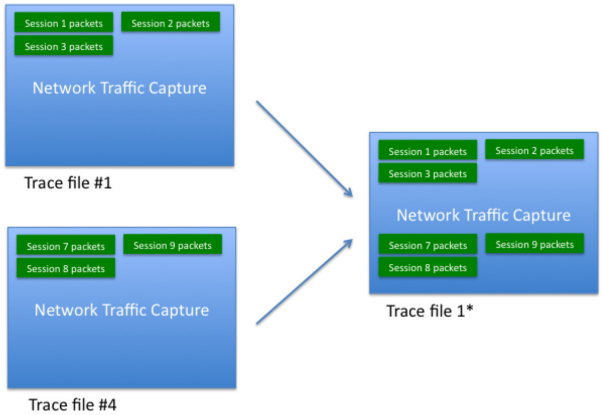
Step 1 - Statistical feature extraction



Step 2 - Clustering points on an HMM Manifold



Step 3 - Discover groups of statistically-similar network hosts



Step 4 - Merge traffic between statistically-similar hosts

Figure 8: IMPACT: Step-by-step operations.

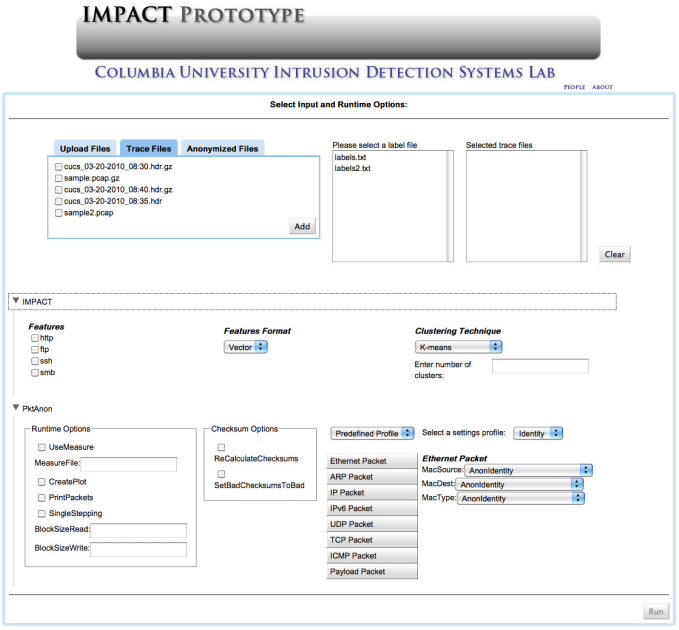


Figure 9: IMPACT interface: showing integration with `PktAnon` framework

Machine Learning: ECML 2007, J. Kok, J. Koronacki, R. Mantaras, S. Matwin, D. Mladenic, and A. Skowron, Eds., vol. 4701 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2007, pp. 164–175.

[15] KRUSKAL, J., AND WISH, M. *Multidimensional Scaling*. Sage, 1978.

[16] MINDROT.ORG. softflowd, March 2011. http://www.mindrot.org/projects/softflowd/.

[17] MINSHALL, G. TCPdpriv. http://ita.ee.lbl.gov/html/contrib/tcpdpriv.html, October 2005.

[18] MIRKOVIC, J. Privacy-safe network trace sharing via secure queries. In *Proceedings of the 1st ACM workshop on Network data anonymization* (2008).

[19] NG, A., JORDAN, M., AND WEISS, Y. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems* (2001).

[20] PANG, R., ALLMAN, M., BENNETT, M., LEE, J., AND PAXSON, V. A first look at modern enterprise traffic. In *Proceedings of the* 5^{th} *ACM SIGCOMM conference on Internet Measurement* (2005).

[21] PANG, R., ALLMAN, M., PAXSON, V., AND LEE, J. The devil and packet trace anonymization. *SIGCOMM Comput. Commun. Rev. 36* (January 2006), 29–38.

[22] PARATE, A., AND MIKLAU, G. A framework for utility-driven network trace anonymization. Tech. rep., University of Massachusetts, Amherst, 2008.

[23] PARATE, A., AND MIKLAU, G. A framework for safely publishing communication traces. In *Proceeding of the* 18^{th} *ACM conference on Information and knowledge management* (2009).

[24] RABINER, L. R. A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of the IEEE* (1989), pp. 257–286.

[25] SPIRENT. Spirent network traffic generator. http://www.spirent.com/, 2011.

[26] WEINBERGER, K. Q., AND SAUL, L. K. Unsupervised learning of image manifolds by semidefinite programming. *Int. J. Comput. Vision 70* (October 2006), 77–90.

[27] WIKIPEDIA. The Stakkato Intrusions. http://en.wikipedia.org/wiki/Stakkato, 2004.

[28] WRIGHT, C. V., COULL, S. E., AND MONROSE, F. Traffic morphing: An efficient defense against statistical traffic analysis. In *Proceedings of the Network and Distributed Security Symposium - NDSS '09* (February 2009), IEEE, IEEE.

[29] WRIGHT, C. V., MONROSE, F., AND MASSON, G. M. On inferring application protocol behaviors in encrypted network traffic. *Journal of Machine Learning Research* (2006).

[30] YIN, J., AND YANG, Q. Integrating hidden markov models and spectral analysis for sensory time series clustering. In *Proceedings of the Fifth IEEE International Conference on Data Mining* (Washington, DC, USA, 2005), ICDM '05, IEEE Computer Society, pp. 506–513.

[31] YURCIK, W., WOOLAM, C., HELLINGS, G., KHAN, L., AND THURAISINGHAM, B. Scrub-tcpdump: A multi-level packet anonymizer demonstrating privacy/analysis tradeoffs. In *Third International Conference on Security and Privacy in Communications Networks (SecureComm)* (2007).