A Behavior-based Approach Towards Statistics-Preserving Network Trace Anonymization

Yingbo Song

Submitted in partial fulfillment of the
requirements for the degree
of Doctor of Philosophy
in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2012

Abstract

A Behavior-based Approach Towards Statistics-Preserving Network Trace Anonymization

Yingbo Song

In modern network measurement research, there exists a clear and demonstrable need for open sharing of large-scale network traffic datasets between organizations. Beyond network measurement, many security-related fields, such as those focused on detecting new exploits or worm outbreaks, stand to benefit given the ability to easily correlate information between several different sources. Currently, the primary factor limiting such sharing is the risk of disclosing private information. While prior anonymization work has focused on traffic content, analysis based on statistical behavior patterns within network traffic has, so far, been under-explored. This thesis proposes a new behavior-based approach towards network trace source-anonymization, motivated by the concept of anonymity-by-crowds, and conditioned on the statistical similarity in host behavior. Novel time-series models for network traffic and kernel metrics for similarity are derived, and the problem is framed such that anonymity and statistics-preservation are congruent objectives in an unsupervised-learning problem. Source-anonymity is connected directly to the group size and homogeneity under this approach, and metrics for these properties are derived. Optimal segmentation of the population into anonymized groups is approximated with a graph-partitioning problem where maximization of this anonymity metric is an intrinsic property of the solution. Algorithms that guarantee a minimum anonymity-set size are presented, as well as novel techniques for behavior visualization and compression. Empirical evaluations on a range of network traffic datasets show significant advantages in both accuracy and runtime over similar solutions.

Table of Contents

1 Introduction **1**

1.1 The need for network trace anonymization 2

1.2 Contributions of this thesis 4

1.3 Thesis outline 7

2 Background and Related Works **8**

2.1 The nature of network trace data 9

2.2 Source anonymization 11

2.3 Network packet traces vs. microdata 13

2.4 Paradigms for network data sharing and anonymization 16

2.5 Measuring anonymity and utility 20

2.6 Attacks and other challenges 22

3 Problem Formalization and Solution Model **26**

3.1 The duality of identity and anonymity 27

3.2 Threat model 28

3.3 Scope of this thesis 30

3.4 Behavior-based traffic-mixture for anonymization 32

3.5 Fundamental assumptions 33

3.6 Privacy and utility 35

3.7 Evaluation methodology 37

4 Statistical Model for Network Behavior **39**

4.1 Feature extraction 41

4.2 Volume distributions modeling 46

4.3 Time-series extension with hidden Markov models 48

4.4 The Protocol-transition Markov model 53

4.5 Technical and implementation details 61

4.6 Evaluations 62

5 Kernel Extensions for Network Behavior Model **68**

5.1 Paradigms in time-series data clustering 70

5.2 Derivation of the probability product kernel for the PTM 72

5.3 Spectral algorithms for PTM-measure host behavior 78

5.4 Accuracy and runtime evaluations 84

5.5 Behavior visualization and low dimensional embedding 88

5.6 Kernel machines and density estimation 92

6 Network Traffic Synthesis and Simulation **100**

6.1 Synthesizing flow-layer statistical samples 101

6.2 Behavior-based traffic interpolation 107

6.3 Behavior-based traffic regression 110

6.4 Passive network-device fingerprints 112

6.5 Synthesizing TCP-layer packet traffic 113

6.6 Time-driven traffic synthesis 116

6.7 Runtime evaluation 118

7 Network Trace Anonymization **121**

7.1 Anonymity via crowd of similarly-behaving peers 122

7.2 Stability of the algorithm 127

7.3 Self-similarity and behavior shaping 137

7.4 Technical details of implementation 144

7.5 Preservation of statistical properties 147

7.6 Connection to existing anonymity metrics 157
7.7 Information leakage and potential attacks 159

8 Limitations, extensions, and future work 164
8.1 Limitations of the behavior-based approach 165
8.2 Incorporating the network topology in the feature set 167
8.3 Anonymity via model-exchange 170
8.4 Behavior-driven anonymous VPN 171

9 Conclusions 173

I Appendices 176
.1 p0f signature list 177

II Bibliography 185

Bibliography 186

List of Figures

3.1 Anonymization paradigms are shown: original traces shown in solid-black, new traces shown in dashed-blue. 32

4.1 Log-scale plot of histograms for traffic-volume by ports. $x < 0$ range represents Received-On ports, $x \geq 0$ represents Sent-To ports. Notice the distinct behavioral patterns between the user and server machines. 42
4.2 Distribution of packet sizes during two hours worth of traffic, randomly sampled over a period of 20 days. 43
4.3 Activity-grid representation: 288 five-minute port histograms; a time-series feature representation for network traffic. 44
4.4 Graphical representation of the hidden Markov model. Shaded nodes represent observable emissions. Unshaded nodes represent hidden states. 48
4.5 Graphical representation of the latent-state flow-transition model. 55
4.6 Graphical representation of the reduced fully-observed protocol-transition Markov model. 57
4.7 Conceptual diagram of the PTM. 58

5.1 One step in the Junction Tree algorithm. 75
5.2 Results of spectral clustering with 10 clusters. 80
5.3 Results of spectral partitioning. The minimum cluster size was set to 5. . . 83
5.4 Runtime scaling comparison of top two kernel-based clustering methods: mutual-information kernel vs probability-product kernel. Dashed line: Mutual Information kernel, solid line: PPK. 87

5.5 Nearest-neighbor clusters are recovered with the CUCS dataset using the PPK. 87
5.6 Kernel-PCA embeddings for CUCS dataset using HMM models. 89
5.7 Kernel-PCA embeddings for CUCS dataset using PTM models. 90
5.8 4-D Behavior embedding. Time step 1. 93
5.9 4-D Behavior embedding. Time step 2. 93
5.10 4-D Behavior embedding. Time step 3. 94
5.11 4-D Behavior embedding. Time step 4. 94
5.12 4-D Behavior embedding. Time step 5. 95
5.13 4-D Behavior embedding. Time step 6. 95

6.1 Behavior interpolation: Two clusters of behavior profiles are embedded ("sos2" and "web3") along with a new class of behavior ("S") that interpolates between these two models. 109
6.2 Filling in missing data: (Left) shaded entries represents missing data. (Right) synthesized entries. 111
6.3 Programming with the Flow-to-PCAP TCP-session-synthesis library. 114
6.4 Synthesized packets are sessionizeable and passes all checks in Wireshark. . 115
6.5 Translation between statistical representation to PCAP. 115
6.6 SYNTHESIZE-NETFLOW maintain $O(n)$ growth with respect to number of flows. Large traffic set with 20,000 flow entries took less than five seconds to synthesize. 118
6.7 SYNTHESIZE-NETFLOW runtime is largely independent of the model size. . . 119

7.1 PTMs are trained for each host within a network trace file in steps 1 and 2. 123
7.2 Time-series models can be conceptualized as points in a time-series space. . 124
7.3 These behavior models are conceptualized as points within a graph where edge weight is solved by the kernel affinity calculated with the PPK. The optimal normalized cut is recovered using a spectral approximation algorithm. 124
7.4 Clusters can of similar behavior are recognizable in this space using the probability product kernel. 125

7.5 Similar hosts are grouped together to form clusters for statistics-preserving crowd-based anonymization. 125
7.6 Anonymity by crowds is actualized through a packet/flow merging transformation on the appropriate hosts in the trace file. 126
7.7 Mixing traffic amongst similarly behaving hosts. 126
7.8 Measuring log-likelihood loss as a function of cluster size (k). 129
7.9 Agglomerative clustering to guarantee minimum cluster size ($k = 5$.) Distributions of cluster sizes at each step is shown. 133
7.10 Results of spectral clustering with agglomerative merging of the resulting clusters. The minimum cluster size was set to 5. 134
7.11 Average packet-count disparity vs. number of clusters in Columbia dataset. 136
7.12 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. 137
7.13 Self-anomaly of different hosts across time. Calculated as normalized negative log-likelihood score of data within a sliding window swept across the traffic. 139
7.14 Traffic smoothing: self-anomalies – segments of traffic with anomaly scores above 1 standard-deviation from the mean are removed. Original is shown on the left, smoothed traffic is shown on the right. 141
7.15 Traffic shaping: inserting anomalies – segments of traffic are modified to induce self-anomalies. Anomalies are inserted at the half-way mark of each traffic set. 143
7.16 Preservation of the traffic-volume distribution. 150
7.17 Preservation of the shape of the unique-ports distribution. (a) 82 hosts are shown in the original traffic, (b) 20 pseudo-hosts are shown in the merged traffic. 151
7.18 Preservation of statistical information is controllable. (a) shows the embedding of similarly behaving hosts and (b) shows the same dataset after a whitening transform. 154

7.19 Visual example of the cover-traffic problem. (a) three similar signals are displayed (b) top plot shows the three signals merged, middle plot shows the aggregate effect, bottom plot shows the problem when traffic sequences are not properly overlaid. 160
7.20 p0f signature format for passive OS fingerprinting. 161

8.1 Small-world-graph connection patterns for a set of hosts on the Columbia network. 168
8.2 Peer cardinality histograms. 1000 hosts on the CU network, sampled over 4 hours across 20 days. 169

List of Tables

4.1 Classification performance of different HMMs on CUCS 80 vs. 80 dataset at the *day*-granularity. Dashed entries indicate infeasible settings. "G-Vec." refers to a Gaussian vector model. 52

4.2 Likehood-evaluation-based classification accuracy of comparable models across datasets. 64

4.3 Distribution of model sizes (number of port-bins) per dataset. 65

4.4 Classification performance comparison of comparable time-series models on CUCS traffic. Dashed entries indicate infeasible settings. PTM dramatically outperforms HMMs. 66

5.1 Fast iterative algorithm for the probability product kernel for the PTM. . . 75

5.2 Faster iterative algorithm for the probability product kernel for the PTM using the maximum-a-posteri estimate. $T = 10$ is a good setting for convergence. 77

5.3 Clustering accuracy comparison between samples from classes of different behavior. 85

5.4 Average runtimes of time-series clustering algorithms. Includes model-training time. 86

5.5 PT-SVM classification accuracy for inter-class behavior; all classes were linearly separable. Representative of between-class classification results. 97

5.6 PT-SVM classification accuracy for intra-class behavior. Web[1,2,3,4] are NAT'd hosts supporting a load-balanced web-server. 98

6.1 Log-likelihood evaluation of genuine, synthetic, and randomly-sampled data across datasets (average of 10 trials.) 106

7.1 Anomaly detection in original and merged dataset. 152

7.2 Ranked anomalies in original and merged datasets. 153

7.3 Distribution of model sizes per dataset, in terms of number of port bins used. 156

Acknowledgments

This thesis was made possible by contributions from several individuals involved in the DHS-funded project on network-trace anonymization. Listed in no particular order, they include Bert Huang, Blake Shaw, Eli Brosh, Kyung-Wook Hwang, Oladapo Attibe, Mariana Raykova, Professors Dan Rubenstein, Steve Bellovin, Tal Malkin, Vishal Misra, and my advisors Tony Jebara and Salvatore Stolfo. The contributions and input from these individuals were instrumental in shaping the foundation and direction of this work. I would also like to acknowledge Steve Gossin, Daniel Robertson, Tim Maddrey, and Dr. Robert Gray of BAE Systems for working with us to develop this technology for future deployment.

During my years at Columbia, I was fortunate to have had the opportunity to interact with many talented individuals that greatly influenced my own development as a researcher. My interests in security arose from my positive experiences in the Operating Systems and Network Security courses taught by Professor Angelos Keromytis, who gave me my start in security research, and introduced me to his talented students Michael Locasto and Angelos Stavrou, who guided me through the whirlwind of my first publication. I am thankful for the great discussions with- and helpful advice that I had received from friends and colleagues of past and present. In no particular order, I would like to thank Kapil Thadani, Pratap Prabhu, Andrew Howard, Delbert Dueck, Stuart Andrews, Gabriela Ciocarlie, Wei-Jen Li, Vanessa Frias-Martinez, Malek Ben Salem, Brian Bowen, Sambuddho Chakravarty, Ang Cui, and Nathaniel Boggs, for the time and energy that they have given me.

I owe my thanks to the Columbia Brazilian Jiu-Jitsu and Taekwondo clubs for providing me with much-needed distractions outside of the office. I have had the privilege to train with great instructors Andrius Schmid and Jason Yang, who, combined, have volunteered over a decade of committed service to Columbia's Jiu-Jitsu program. They are great ambassadors for our sport and their Renzo Gracie lineage.

Finally, I would like to again thank my advisors; Professors Tony Jebara, for sharing with me his extensive knowledge of Machine Learning, whose work has been influential in my own research, and Salvatore (Sal) Stolfo, for his reliable support and advice over the years. Sal's hard work in keeping our lab continuously well funded, even through episodes of failing personal health, has afforded us great opportunities and freedoms to cut our own paths as researchers, and we, his students, will always be grateful for that.

For my family.

Chapter 1

Introduction

In the year 2004, a hacker known as "Stakkato" broke into Terragrid, a large world-distributed research computing platform. The attacks spanned a total of 19 months, and resulted in the successful infiltration of thousands of university, corporate, and military machines, in both the United States 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 effected organizations to easily share data amongst each other. This was due to the fact that, among the attack-fingerprints that forensic experts would want to extract from this traffic, such network data also contained private, sensitive communications information which could not be released without harm to the individual organizations – this data was not easily removable, at the time.

This thesis proposes a novel technique for enterprise-scale network packet-trace source anonymization, motivated by behavior-based crowd-anonymity. The methodology describes herein is designed to provide source-anonymity for individual host identities while minimizing the information-loss within the aggregate network characteristics. The primary motivation for this work is to facilitate the distribution of anonymized network traffic between organizations without risk of privacy-loss.

1.1 The need for network trace anonymization

Stakkato's attack pattern is a familiar one in the security community: a vulnerability is discovered in a service, potential targets are scouted through scans and probes, vulnerable targets are exploited and subsequently used as a platform to identify and exploit further targets. Recovering the logs of such events is therefore of great interest to the security community, in providing insights into where failures in security – whether technical or procedural – may reside, especially if the same attack span heterogeneous networks.

Beyond the forensic-security setting, 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, and the network research community is no exception. It is because of this need that organizations such as openpackets.org [Bejtlich *et al.*, 2011], and the more recent U.S. Department of Homeland Security-sponsored predict.org [PRE-

DICT, 2011], 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 [Mirkovic, 2008]. This result, along with other published opinions of a similar nature [Alllman and Paxson, 2007], reflect a deficiency in the current network and security research fields of publicly available large traffic-capture datasets.

In modern network research, there exists a clear and demonstrable need for open sharing of large-scale network traffic datasets between research organizations. Many security-related research fields, such as those focused on detecting exploits, DDoS attacks, or worm outbreaks, would stand to benefit given 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, have been relatively limited. 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 certain research goals, such as reproducibility-of-results, difficult to achieve.

Network Data Anonymization (NDA) is emerging as a field that is dedicated to this

problem, with its first workshop organized in 2008 [NDA, 2008]. The predominant direction in NDA is content removal and masking, which includes deletion of packet payloads and masking of packet 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` [Pang *et al.*, 2006], which is a policy-driven framework for utilizing a range of such transformations. Tools such as these facilitate the removal of human-identified signatures that might fingerprint users, hosts, or services that should otherwise remain anonymous.

However, recent research has demonstrated that beyond superficially observable datums such as usernames and 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 of the traffic traces are encrypted, HMMs were still able to reliably isolate these protocols from the encrypted dataset [Wright *et al.*, 2006]. This examples, and others discussed in § 2.6, highlight certain shortcomings in current anonymization frameworks – in particular, ignoring the statistical idiosyncrasies of network protocol-, application-, and user-behavior. As the field of network traffic anonymization progress, is certain that behavioral fingerprints should receive a more prominent role.

1.2 Contributions of this thesis

The purpose of this thesis is to develop a framework for behavior-based source-anonymization for hosts within offline network trace data, in an approach that follows the mixnet principle of anonymity-by-crowds. This is achieved through time-series modeling of host behavior conditioned on their traffic samples, and matching hosts with similar behavior. Anonymization is achieved partially through replacement of individual identities with group-identities. Group assignment is based on the similarity of hosts with other members. In addition, the methodology for statistics-driven traffic perturbation and shaping is demonstrated, as well as the methodology for synthesizing data for targeted behaviors. The intermixing (or "clustering") of host traffic effectively anonymizes each member of that cluster (pro-

viding k-anonymity) while simultaneously preserving, in the aggregate, the statistics that characterize the members of that group. Other approaches to traffic-mixing without such constraints naturally causes information-loss in the aggregate statistics. However, behavior-based matching allows us to drive these transformations such that information-dilution is minimized. As this process is independent of other anonymization transforms, it is compatible with existing frameworks [Pang *et al.*, 2006; Gamer *et al.*, 2008] and can be conceptualized as an additional layer in existing anonymization chains.

Our techniques follow a kernel-based approach to learning and therefore extend naturally into a range of learning algorithms. These include derivations of support vector machine (SVM)-like classifiers, kernel-based data compression and visualization techniques, as well as incorporation of measurements from different feature sets using kernel combinations. Primarily, the kernel approach allows us to utilize graph-partition methods that are robust when the underlying distribution of the dataset does not conform to easily parameterized models, as is often the case with network-behavior. A more exact enumeration of the contributions of this thesis is as follows:

1. **Time-series model for network behavior** A new and efficient time-series model for host behavior is presented; one that is trainable using high-level flow-layer meta-data, and whose training and likelihood-evaluation times maintain linear growth in the number of data samples.

2. **Kernel metric for model-similarity** A new kernel, motivated by the concept of probability product kernels, is derived for comparing similarity between models in an efficient and scalable manner. This proposed kernel is efficient in both runtime and memory costs, requiring orders of magnitude less computation than the closest competitors. An approach for incorporating other similarity metrics, such as those based on network structure, is provided.

3. **Clustering and partitioning of models** New algorithms for clustering host behavior are derived, based on graph-partition methods. The spectral algorithms are shown to outperform comparable parametric methods in the same category. Dramatic improvements in both accuracy and runtime over existing methods are demonstrated.

4. **Data compression and visualization of behavior** Kernel principal component analysis (KPCA) and similar data compression and embedding techniques extend naturally from our model. We propose new methods of network behavior visualization based on these techniques and demonstrate their utility.

5. **Data synthesis with targeted behavior** Algorithms for sampling network traffic from our models, according to targeted behavior, are proposed. Behavior-interpolation, regression, and shaping are presented, as well as the techniques for translating statistical samples back into packet trace data. This leads to potentially new avenues for privacy-driven network data sharing.

6. **Network traffic anonymization** These previously described methods are combined to form a framework for behavior based network trace anonymization. The stability and accuracy of this approach is demonstrated and connections to existing anonymization theory are shown as we show how anonymization requirements may be inferred from the data.

While data anonymization has been a topic of research for several decades, prior approaches have focused on very restricted domains such as statistical databases, referred to as "microdata" and it is in this area were most progress has been made [Dwork *et al.*, 2010; Dwork, 2006; McSherry and Mahajan, 2010; Kifer and Machanavajjhala, 2011; Kelly *et al.*, 2008]. Network traffic datasets, such as packet or netflow captures, represent a fundamentally different challenge – containing information that is both unfiltered and multifaceted. The context of source-anonymization which we study is a comparably new field, and have focused largely on policy-driven frameworks with sets of obfuscation primitives [Gamer *et al.*, 2008; Mogul and Arlitt, 2006; Minshall, 2005; Blanton, 2008; Pang *et al.*, 2006]. Online source-anonymization has been researched in the concept of mix-nets and onion routing [Dingledine *et al.*, 2004; Freedman and Morris, 2002; Syverson *et al.*, 2001].

Unlike micro-data, network trace anonymization does not have a significant focus on reducing statistical information-loss. Our proposed approach applies the intuitive concept of anonymity-by-crowds to offline traffic, while reducing information-loss by selectively mixing

traffic with similar behavior characteristics, measurable by novel machine learning methods. Our transformations are compatible with related obfuscation primitives found in similar frameworks, and is driven by the same statistics-preservation motivation as those found in microdata anonymization. As such, our methods draw from the best aspects from all of these fields to produce a new approach towards network-trace source anonymization.

1.3 Thesis outline

This thesis is distributed across eight chapters. The background and related work for network trace anonymization is presented in Chapter 2. Chapter 3 provides a formalization of the threat model along with our proposed solution, and specifies the parameters and scope of this research. The main components of our solution are independently described in Chapters 4 (behavior modeling), 5 (similarity measurement) and 6 (traffic synthesis). Chapter 7 (anonymization) shows how these previously described components fit together in a behavior-based source-anonymization system. Discussions on the limitations of our approach and how these are address, along with future work are presented in Chapter 8.

Chapter 2

Background and Related Works

Introduction

This chapters provides a more detailed summary of related works in the field of network data anonymization (NDA). While "anonymization" is a broad term, the citations contained herein focuses on topics related specifically to network-trace source-anonymization. The chapter begins with a description of source-anonymization and the type of network data we focus on. Further citations include works on existing paradigms in network-trace anonymization, utility and privacy measurements, de-anonymization attacks, and other related challenges. Citations and comparisons with microdata anonymization are included to provide a comparison of similarities and differences between these two related fields. To make the distinction, where appropriate in this chapter, we use the term "sanitization" to refer to the process of removing sensitive (potentially privacy-piercing) artifacts from network traffic. This process typically entails modifications of packet headers and payload contents. We the term "anonymization" in a more broader sense to refer to the process of erasing individual host identities from the network trace. Sanitization can be a component of an anonymization methodology.

2.1 The nature of network trace data

Our study focuses on enterprise-level network environments and packet-capture (PCAP) data as well as Netflow data. PCAP data is assumed to be captured at the network scale, captured through a SPAN port at a border gateway. Packet payloads are not required as our techniques do not make use of content information, however we expect the layer-3 packet headers to be mostly complete and unmodified – though IP addresses may be obfuscated. Our methodology utilizes statistical features on network connections, therefore the same techniques apply to Netflow data as well, since it naturally encapsulate this information. Given that such data represents an overview of a network's communication structure, it represents many different categories of behavior – some of which may be independent of one another – thus, it is necessary to delineate tiers of behavior. Particularly, we breakdown traffic behavior into the following layers:

1. **Protocol** This refers to the data-transfer protocol layer (*i.e. HTTP, SSH,....* This layer encompasses such features as: size, frequency, and timing of packet exchanges. Behavioral characteristics at this layer is implementation-driven and not influenced by user actions. For example, the negotiation of TCP window sizes is operating-system driven, and not determined by the user per connection. Characteristics such as packet sizes, timings, port values, and measurements based on re-transmission are available at this layer.

2. **Application** The application behavior layer is measured by the patterns within the protocol-layer. An application may use different sets of protocols, in different patterns. An example of this is a web browser, which may initiate invoke both HTTP (port 80) and HTTPS (port 443) connections. This layer encompasses details such as type and frequency of protocols used. One characteristic of web browsers, for example, is the typical behavior of sending a single small outbound connection which represents the web request, followed by a larger incoming connection to the same ephemeral port which represents the response from the web server.

3. **User/Host** The host layer is measured as a collection of application behaviors. This layer represents the behavior of machines and is closely related to the roles that such machines fulfill. The characteristics of behavior at this layer incorporate usage statistics such as time of use, frequency, and duration. Users, network daemons, and other devices, are identified and distinguished amongst each other at this level.

4. **Network** The network layer behavior incorporates a collection of user/host layer behaviors profiles and defines the characters for a network as a whole. This includes additional information such as the distribution of hosts on the network, the overall distribution of services, protocols, as well as the layer-2 infrastructure used (Ethernet, ATM) and network topology. Identities at this layer attribute the dataset to specific providers such as businesses, universities, and government networks. This layer is considered in the context of providing provider anonymity.

A more exact specification of packet trace characteristics in large enterprise networks can be found in related studies [Pang *et al.*, 2005; 2006]. Previous work have demonstrated

classification results on each of these layers individually. This includes Wright *et al.* [Wright *et al.*, 2006] in demonstrating that protocol-level behavior is measurably distinct, and that individual protocols such as HTTP and FTP can be uniquely identified even when the contents of these channels are encrypted. Role-based prediction based on isomorphism in the traffic dispersion graph [Iliofotou *et al.*, 2007] has been studied by in the context of identifying similar types of servers within a network [Tan *et al.*, 2003]. Similar works such as identifying communities of interest within a network focuses specifically on unsupervised learning based on application-layer characteristics. Host-behavior profiling has also been studied in the context of identifying network profiles based on the distribution of services [Dewaele *et al.*, 2010]. These citations focus primarily on those works that have a focus on role-identification and prediction. Specialized network measurements such as modeling hosts or networks for anomaly detection are numerous, their enumeration is omitted due to relevant; these techniques typically focus on specific aspects of the application-layer behavior and are not designed to generalize to the context of host-layer behavior.

2.2 Source anonymization

This thesis focuses on the problem of offline source-anonymization for network traces. More specifically, we focus on layer-3 (Internet layer) host anonymization, where the principle entity is any device with an IP address that can send and receive layer-3 traffic. In this setting, the IP address is the primary – though not exclusive – identifier (or "label") for each device.

One of the earliest examples of source anonymization goes back to 1981, when Chaum proposed the concept of a "mixnet" [Chaum, 1981]. In a mixnet, a user's traffic is routed through a network of multiple independent nodes, where it is mixed among traffic from other users, before ultimately exiting the network from a set of specific "exit nodes" to reach its destination. From the perspective of the end point, all users of the mixnet would appear to originate from the same set of exit nodes, regardless from where they truly originate. Thus, the system provides anonymity by hiding the true origin of the traffic from the end point. Anonymous web proxies such as `anonymouse.org` prevent the end point – and any

entity listening in between the two end-points – from identifying the source origin of web-site requests. Currently, in the year 2012, this technology is often used in countries where Internet filtering is performed, to allow dissidents to access foreign websites which would normally be blocked by their government. Tor [Dingledine *et al.*, 2004; Tor, 2011] is the most well-known example of a modern mixnet implementation. A user's traffic (not restricted to any particular protocol) enters the Tor network and exits through a set of specific exit nodes. Internally, Tor establishes a circuit between a set of independent pass-through nodes to further aid in masking the origin of the traffic. A virtual private network (VPN) can serve the same purpose. Many organizations allow individuals to connect to a VPN in order to access protected internal networks, and while anonymity might not be the main motivation, all Internet-bound traffic sent through a VPN will also exit through a set of VPN exit nodes, thus providing a similar anonymization transformation.

Mixnets typically share two common design goals: 1) they mask the IP of the originator and 2) they provide cover-traffic by way of traffic mixture, by aggregating connections through a set of common exit nodes. The second point is particularly relevant to this thesis as it demonstrates the strategy of "anonymity by crowds." The motivation behind this is intuitive: if the source and end point connections were always distinct one-to-one pairings, then if one segment of traffic is successfully de-anonymized, all traffic from that session can be attributed back to a particular origin without ambiguity. Further, without mixing traffic from multiple sources at the exit nodes, de-anonymization is essentially the problem of traffic-matching, and attacks, such as timing analysis, can be used to infer the originator; the traffic from a particular origin might be the only activity leaving an exit-node, therefore it is not challenging to match the source and destination.

Mixnets, such as Tor, are widely-used and reliable methods of anonymization. However, these are *real-time* solutions for live traffic. Analogous systems, based on this mixnet principle, currently do not exist for anonymization of pre-captured network traffic. The main focus of this thesis is to apply this concept of anonymity-by-crowds to develop a obfuscation methodology that would allow existing datasets to be similarly anonymized.

2.2.1 Pitfalls of synthetic data

Use of synthetic data for network trace anonymization has been relatively unexplored. The reason for this is that generating realistic and useful synthetic 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 badly. One example is the 1998 DARPA KDD dataset which was synthetically generated to test intrusion detection systems. 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. It is unknown how many methods might have unwittingly locked onto this unintentional hidden label, 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 [Brugger, 2007]. While this is an extreme example such problems, it nevertheless highlights a pitfall in this particular approach.

Synthetic generation of large datasets is not an alien concept – many well known traffic synthesizers exist, including very powerful enterprise-level generators provided by companies such as BreakingPoint [BreakingPoint Systems, 2011] and Spirent [Spirent, 2011]. These solutions typically serve purposes such as testing loading-balancing, routing, or firewall systems. In these settings, low emphasis is placed on achieving "realism" in the dataset – it matters less that these traffic streams were not generated by human operators, or capture/preserve actual security incidents.

This thesis describes one of the first machine learning driven techniques for synthesizing network traffic with measurable accuracy.

2.3 Network packet traces vs. microdata

Currently, much of the progress in information-theory-based data-anonymization has been undertaken in the domain of *microdata* anonymization research. "Microdata" is a term used to describe information that is presented in the form of statistical tables. This domain include examples such as patient surveys in medicine, where each row may represent a patient and each column a particular disease – the entries being one or zero, representing

whether a patient had that disease or not. This would allow computations such as total number of patients with a particular disease, statistical means and standard deviations, and other related measurements. Another example would be United States census data, where each row may represent a particular household and each column represents a certain census metric such as family income, or number of people in the house. This section describes the main similarity and differences between network packet traces and microdata.

Many successful anonymization paradigms have emerged based on studies of microdata. Differential privacy [Dwork, 2006], for example, is regarded as ones of the most well-developed measurement for microdata anonymity. This theory presents many attractive features such as bounds for privacy risk. It is relevant to wonder why these theories do not naturally carry over to the network traffic anonymization as well. The reason for this is simply because the two classes of information are fundamentally very different. The distinguishing properties of microdata are that it is both *processed* and *structured*. We refer to this data as "processed" in the sense that a human operator, or human-designed system, took the collected unfiltered response and extracted the useful information from the raw data, further and summarized this information into measurable quantities. For example, a human census surveyor would take the given response "there are five members of this house-hold" and enter "5" into the corresponding entry in the table. In this case, the number of house-hold members is the pre-determined unit of measurement, and the (often difficult) challenge of "feature extraction" is being performed by the human operator.

We refer to the data as "structured" in the sense that the boundaries of the information are constrained by the nature of the statistical representation, and are well defined in this representation. For example, given two sentences "five people live in this house-hold" and "there are two persons living here," the surveyor would enter "5" into the entry for the first house-hold and "4" for the second. When one wishes to compute which house-hold held more members, one does not compare the first sentence to the second directly and perform a language processing step to infer the semantic meaning of each phrase; rather, the extracted statistical features are compared. The meaning of each element of the table is similarly well-defined, and the types of comparisons undertaken on the table elements (minimum, maximum, means, standard deviations, *etc*) are intuitive.

Network trace data, on the other hand, exists in a significantly different format than the prior example. "Network trace" is a term typically used to describe packet-capture data – snapshots of raw packet-level network activity – typically retrieved from tools such as Tcpdump [tcp, 2011]. This reference may also include other data, such as Cisco Netflow records. In this thesis, we make the distinction explicitly, and use "network trace" to refer specifically to packet data, and refer to Netflow by this name directly.

In contrast to microdata, the form of network-trace data examined in this thesis (Netflow and packet-capture (PCAP)) do not have intuitive mappings into a mathematical space – in the sense that one cannot intuitively measure the difference between two packets, or – what is a more fundamental challenge – even establish intuitive units of measurement for a collection of data. The challenge of measurement is fundamental: can a packet be considered a basic unit of measurement? can a session? In the same sense that the meaning of different sentences cannot be measured by the similarity of individual words, there are no definitive methods to measure the similarity of two network sessions based on comparisons made between the individual packets that comprise these sessions. Furthermore, trace data represents a low-level view of a multi-tiered communication medium, *i.e.* the TCP/IP stack. Transmission Control Protocol (TCP) is a state-full, full-duplex channel that is designed to carry information over the state-less datagram routing Internet Protocol (IP) layer. Application-layer data (the "content" of the TCP session) is broken into smaller individual datagrams of varying sizes during transport. In practice, these datagrams emerge across networks as "packets," with maximum sizes negotiated based on the path's maximum transmission unit (Ethernet, for example, has a default MTU of of 1500 bytes.)

In general, network data is different from microdata in that it is inherently multifaceted, and encompasses the raw information of the network environment without prior feature extraction. It is not always possible to enumerate the desired features from the network data *a priori*. An obvious example of this would be forensic-analysis environments, where the security researcher might not know the signatures of the exploit being investigated, thus it is not possible to enumerate the desired measurements ahead of time. Coull *et al.* provides a general summary of the differences between these two types of data as well as their respective anonymization paradigms [Coull *et al.*, 2009].

Though microdata represents a different domain of information, some of the theoretical results from this field are relevant and can be applied to network-trace anonymization. These measurements such as k-anonymity, anonymity set size, l-diversity, *etc*, are described in later sections within this chapter.

2.4 Paradigms for network data sharing and anonymization

Existing network-trace-anonymization techniques typically fall into one of two paradigms. The first is the class of query-based systems which control the data on a protected server and allow users to make restricted queries. The second is based on utilizing one-way transformations on the original data to sanitize it of sensitive information. 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 where the value of the data is not so easily broken down and quantified into independent datums (such as statistical 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 important of policy-driven systems, and concludes with some discussion on relevant attacks against modern anonymization systems.

2.4.1 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 form 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. This representation is fundamentally congruent with microdata assumptions. For example, one may ask for the count, mean, or standard deviation for a particular entry in set of records; or they may ask for the range of a field, or the size of a set, whether or not a value is

set in a packet header, *etc.* More quantifiers allow greater flexibility in processing; though, regardless of how expansive this range of queries may be, this paradigm remains depends on a priori specifications of these quantifiers. More subtle is the assumption that the operator fully understands the full value and scope of the data, a priori. As discussed in the case of forensics, the artifact of interests are usually not known until the data is examined. The secure query-based approach proposed by Mirkovic [Mirkovic, 2008] is representative of the query-based paradigm. In this approach, an SQL-like query language is presented to the user, and security policy enforcement is maintained on the server. Parate *et al.* [Parate and Miklau, 2009; 2008] 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 SC2D system is a rather novel approach, the authors call for the exchange of open-sourced feature-extraction algorithms. The data owner runs foreign feature extraction code, and returns on the results to the requesting party [Mogul and Arlitt, 2006].

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. The 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 [Pang *et al.*, 2005]. Hosting all of the query-processing for such a massive dataset in one central location is logistically difficult, if not impossible. The second problem is that the value of the trace data is often not immediately known, especially in the security domain. Often, it is only through careful forensic examination of the data that any insight into utility can be obtained. Values such are signatures for intrusions are hard to define *a priori*. In domains where the value of the information is more clearly defined, though, such as health records, the statistical-database model is more appropriate.

Arguably the strongest benefit to using query-based systems is the availability of strong-privacy guarantees – specifically "Differential Privacy." Differential privacy [Dwork *et al.*, 2010; McSherry and Mahajan, 2010] is an analysis-based anonymization platform where a

set of measurements is specified *a priori*; the measurements of these specified analytics can then be modified in a way that is guaranteed to provide a quantifiable level of anonymity. As a specific example, in McSherry and Mahajan [McSherry and Mahajan, 2010], a set of measurements (worm fingerprinting, stepping-stone detection, *etc*) is specified, and differential privacy serves as an anonymization procedure in a mediated analysis platform where measurement results, not modified data, is return to the user.

2.4.2 Sanitization-based systems

The second paradigm, which is more applicable for network-trace data, relies on applying sets of transformations to the data to sanitize it of sensitive information. This process ideally cleanses the dataset of discriminative identifiers, such as IP addresses, packet-payload content, *etc*, which might pierce privacy when released. For network traffic specifically, sanitization-based approaches is, in our opinion, a less precise but more general approach to anonymization. Sanitization removes potentially privacy-piercing information from the data, ideally in a way that minimizes the information lost. This approach is more practical in network traffic analysis when the data is multifaceted and analytics cannot be specified a prior. Several notable works in this area are worth mentioning; these are listed here in the chronological order of their release. `Tcpdpriv` [Minshall, 2005] is one of the earliest traffic-anonymization tools. A simple packet-header sanitization tool, it was invented before modern de-anonymization attacks were known. It allowed the user to modify certain header values such as the IP address. `Tcpurify` [Blanton, 2008] is another earlier, although slightly more advanced, sanitization tool that was developed at a university, by a professor who wanted to release network data for students to conduct experiments on.

One of the first major influential tools released was `tcpmkpub` by Pang *et al.* [Pang *et al.*, 2006]. This tool presented the first policy-driven anonymization framework, where the user can set from a set of anonymization functions (referred to as "primitives") to use over the data. These functions includes options to remove the payload, hash the header values, obfuscate the IP addresses, and more. It provided transparent handling of the tcp-stack, and advanced protocol-parsing functionality, and was driven by an innovative, customizable anonymization language. Huang *et al.* proposed a system for large-scale network packet

trace sanitization based on presenting exemplars to expert systems [Huang *et al.*, 2011].

The `CryptoPan` was released by Fang *et al.* [Fan *et al.*, 2004] as the first provable prefix-preserving IP anonymization technique. Earlier methods for IP-obfuscation simply performed a randomized one-way one-to-one mapping for each IP address independently. This effect breaks the subnet address structure that IP addresses were built to represent. By preserving a prefix in the transformation, CryptoPan allowed IP obfuscation without erasing the subnet structure in the resulting dataset. `SCRUB-tcpdump` [Yurcik *et al.*, 2007] is a tool that allows one to extract only the useful values from a trace dataset and drop everything else.

Statistical traffic obfuscation techniques such as those presented by Wright *et al.* [Wright *et al.*, 2009] attempts to mask the statistical signature of network protocols. Prior to this work, much of the attention had been focused on content and static signatures within network traffic. The authors demonstrated the feasibility of using statistical signatures to both identify and mask traffic streams. Their technique used padding for packets to smooth statistical signatures from traffic. Deep-packet anonymization is an interesting corner-case solution presented by Foukarakis *et al.* [Foukarakis *et al.*, 2009], this technique examines the content of packets dynamically and uses emulation to unravel polymorphic-encoded shellcode. It then obfuscates the attacker's call-back IP address.

Finally, the most recent advance in network-trace anonymization is the PktAnon framework [Gamer *et al.*, 2008] 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. Their framework uses an XML-driven anonymization-policy language and can handle a broader range of network protocols including IP-in-IP.

> "No perfect anonymization scheme exists and therefore, as in much of the security arena, anonymization of packet traces is about managing risk." Pang *et al.* [Pang *et al.*, 2006]

Policy-driven anonymization frameworks such `tcpmkpub` and `PktAnon` have emerged as the most popular tools in current use. This stems from a 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 information you are aware of and not aware of. 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 powerful 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 [Pang *et al.*, 2006; Alllman and Paxson, 2007; Gamer *et al.*, 2008].

2.5 Measuring anonymity and utility

A number of works have proposed methodologies for evaluating anonymity and the utility of the resulting data, post-anonymization. Many are derived for microdata, and while not all of these measurements are consistent with requirements for network-trace anonymization, the principles behind these metrics are congruent with those of trace anonymization; the most relevant of these methodologies are summarized here.

2.5.1 Measuring anonymity

An overview of general network data anonymity metrics can be found in a survey by Kelly *et al.* [Kelly *et al.*, 2008]. Among these metrics, the ones most relevant to network-trace anonymization include the following generalized metrics: k-anonymity, l-diversity, and t-closeness. Another similar review by Coull *et al.* [Coull *et al.*, 2009] provides a similar survey of metrics.

k-anonymity ties the degree of anonymity with the size of the crowd, such as the number of members in a mixnet. In general, a larger k implies implies more anonymity for each member, as it becomes harder to match the end traffic with the originator; the pool of

potential origins increase, therefore, the probability of obtaining a correct match decreases. k-anonymity is an instance of a generalized measurement of anonymity, where a set of indistinguishable entities, with respect to a specific set of identifiable attributes, is known as the equivalence class of the anonymity set. The size of the equivalence class is positively correlated with the level of anonymity. l-diversity is a data metric that quantifies the uniqueness of a measurable attribute. As an extreme example, the IP address of a network host is a low-entropic data metric, as each host has only a single unchanging unique IP address associated with it. t-closeness is a related metric that is associated with the variance of a distribution. For example, consider the type of network protocol observed in each session. If hosts A and B on a network exclusively exercised protocols HTTP and FTP, respectively, then regardless of how their IPs are obfuscated, it is always possible to identify the origin of each session based on the protocol value: all HTTP traffic belongs to A, and all FTP traffic belongs to B. In this example, A and B appear very different from each other, and are thus distinguishable.

Other, more specialized – though somewhat tangentially related – measures of anonymity also exist. [Zhang *et al.*, 2007] discusses measurements of information-disclosure under realistic assumptions on privacy loss. Assessing a concept of global disclosure risk is discussed in [Truta *et al.*, 2004]. Transparent anonymization studies the properties of thwarting adversaries who know the algorithm [Xiao *et al.*, 2010]. Most progress in quantifying anonymity have been in the more restricted domain of statistical-databases where the query-based model can be well-bounded. By restricting the interaction of the system to input queries and output results, it is possible to derive functions and bounds on the exchanged data, and it is within this domain that more powerful anonymity metrics such as "differential privacy" have been successfully applied to network data anonymization [Dwork *et al.*, 2010; McSherry and Mahajan, 2010].

2.5.2 Measuring utility

General metrics of utility are difficult to enumerate as utility can be a subjective notion. The primary reason for this is the fact that utility and privacy are often incongruent. Perfect preservation of utility entails not modifying the data at all, therefore retaining all

of the original information. Conversely, perfect anonymization involves removing all of the distinguishable data, which, in essence, entails merging all of the traffic together into a single profile representing the statistical mean of all entities. The practical solutions are found in the continuum between these extremes. The optimal measure is different based on the specific utility task: for example, a measurement task of discovering the distribution of heavy-hitters on a network does not require evaluating packet payload content, therefore this element can be removed from the traffic without effecting utility. However, the same anonymization step of removing the packet payload might destroy the utility when the task is forensics oriented, where detections might be based on the presence of specific signatures strings in the payload. Nevertheless, some generalized metrics for utility are relevant for specific case and have been proposed in previous works. Framework for evaluating the utility of data altered to provide confidentiality is discussed in [Karr *et al.*, 2006]. Measuring the utility of certain anonymized data publishing methods is discussed in [Brickell and Shmatikov, 2008]. The risk/utility trade-off of prefix-preserving IP transformations has been studied by Burkhart *et al.* [Burkhart *et al.*, 2008]. The effectiveness of k-anonymity against traffic analysis is discussed in [Hopper and Vasserman, 2006].

From a forensic-security perspective, two papers have been published on evaluation of IDS-alerts pre- and post-anonymization [Yurcik *et al.*, 2007; Lakkaraju and Slagell, 2008]. Their results show that minute details in how anonymization is performed can yield interesting consequences. Lakkaraju & Slagell, for example, discovered that transforming port values to 0 triggered Snort's "BAD_TRAFFIC TCP" rule which, in their experiments, caused a 550× increase in false alerts [Lakkaraju and Slagell, 2008].

2.6 Attacks and other challenges

Many challenges are associated with network-trace anonymization. Primarily, any proposed system need to be mindful of both design requirements, which may limit their utility, as well as de-anonymization attacks; the latter of which encompass a relatively broad surface, as this section will discuss.

Offline network-trace anonymization platforms need to meet certain design requirements;

the most important of these include: transparent handling of the TCP stack, the ability to parse common protocols – or the capability to be easily extended to do so –, the ability to easily filter the traffic to remove artifacts such as port-scans and other overt fingerprints, and – where applicable – secure key-exchange mechanisms. These basic design constraints evolved from a gradual realization of the privacy-piercing attack surface. De-anonymization attacks exists at all layers of the TCP stack: from host-counting through NATs at the internet layer [Bellovin, 2002], to service, protocol, and host identification at network layer [Coull *et al.*, 2007b; Foukarakis *et al.*, 2009], to unmasking communication traffic at the application layer [Coull *et al.*, 2007a], and identification of individual hosts through encrypted channels [Chakravarty *et al.*, 2008].

Two types of attacks can be distinguished: the *passive* attacker infers information from anonymized traces. The *active* attacker injects artifacts into the dataset before anonymization and rediscover these artifacts in the anonymized dataset. De-anonymization attacks can be online and offline: from statistical inference methods [Coull *et al.*, 2007a] to methods that actively interfere with the encrypted channel and measure the resulting jitter to break anonymity [Chakravarty *et al.*, 2008]. A representative selection of such attacks are broken-down in the following paradigms.

Artifact correlation: Passive inference techniques correlates side-channel information to pierce anonymity. For example, one can use port scans to de-anonymize traffic as follows: if one host linearly scans the IP range 128.59.1.1 $\rightarrow$ 128.59.1.255, it would yield an obvious behavioral fingerprint. If the granularity was not perturbed then this signature is observable in the anonymized dataset, in the obfuscated IPs. If the timestamps were not changed, and the attacker was was able to de-ofbuscate the first IP address he would know that the second obfuscated IP observed corresponds to 128.59.1.2, the third to 128.59.1.3, and so on. Further discussed in [Koukis *et al.*, 2006], this can be classified as k-anonymity, or anonymity-set-size, attacks.

Information leakage exploitation: Several fingerprinting techniques take advantage of the entry-diversity weaknesses, or other information leakages to infer information. Bellovin

showed how to infer the number of hosts behind a NAT by counting the id-number entry of packets at the exit point [Bellovin, 2002]. Using the clock-skew of the OS kernel has been used fingerprint remote hosts, as demonstrated by Kohno *et al.* [Kohno *et al.*, 2005]. Certain operating systems, including different versions of the same operating system, will implement the TCP stack differently, and therefore behave slightly differently based on how certain flags are set in packets. Active OS fingerprinting tools such as `NMap` and passive ones such as `p0f` [Zalewski, 2006] utilizes these idiosyncrasies to identify the OS version. More esoteric techniques such as tracking post-exploit host responses to identify vulnerabilities, and even examining the (encrypted) content of exploits to identify attackers and victims have been explored [Foukarakis *et al.*, 2009]. Information leakage is, by far, the most popular method of de-anonymization.

Statistical inference: An emerging class of de-anonymization attacks uses statistical inference to break anonymity. Coull *et al.* [Coull *et al.*, 2007b] modeled the behavioral characteristics of network hosts in the publicly available LBNL trace dataset to identify DNS servers, to a certain degree of success [Alllman and Paxson, 2007]. Wright *et al.* [Wright *et al.*, 2006] modeled certain network protocols and demonstrated that these models were sufficiently discriminative for protocol identification without content information. Modeling the distribution of packet sizes when requesting static web-pages can be used to fingerprint webpages without content, as shown by Koukis *et al.* [Koukis *et al.*, 2006]. In addition, prediction using access-order of content distribution networks was demonstrated by Coull *et al.* [Coull *et al.*, 2007a].

From these results a general theme emerges, that anonymity is mostly a game of exploiting information that you are aware of versus information that you are not aware of – the former is leveraged to preserve the utility of the dataset during the anonymization process, whereas the latter is used to pierce the privacy of the anonymized data.

Concluding remarks

The current state-of-the-art in network-trace anonymization focuses on the approach of sanitization: datasets should begin with captured traffic and, through anonymization, private information should be removed, leaving the remaining information as the resulting output. In one extreme, the optimal anonymization procedure is simply removing all information from the data; conversely, the optimal utility-preserving transformation makes no changes to the dataset. These extremes 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 – different users will apply their own definitions. Privacy therefore, much like the rest of security field, revolves, to a certain extent, on risk-management, 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 for privacy, with minimal dilution of the fidelity of the dataset.

Chapter 3

Problem Formalization and Solution Model

Introduction

This chapter formalizes the problem that we study in this thesis. An overview of the scope of this research is provided. This includes the threat model, the scope of the work, the proposed solution model, the definitions used, and an overview of the methods of evaluation. The need for anonymization is explained in more detail and a more exact definition for source-anonymity is provided. This chapter also covers the fundamental assumptions underlying the foundation of this research.

3.1 The duality of identity and anonymity

This section specifies the operative definition of "anonymity." Though many prior works have been published on network-data anonymization and privacy preserving paradigms (such as differential privacy), few universally applicable definitions exist for anonymity in packet trace anonymization. This is due to the fact that anonymity is often tied with utility (the intended use for the dataset) and is thus considered to be subjective given different environments, and may be conditioned on the protocols and services that a host exercise, and not only hosts themselves – a dataset of web logs may have a different standard for anonymity than a dataset of packet headers used for routing configuration, for example.

In this thesis, identity is considered at the host-level and is tied directly with individual hosts (as opposed to anonymization of protocols or application-use.) It is conditioned on the host's uniquely assigned label *i.e.* the address (IP, Ethernet) of that host on the network. It is directly related to that host's "traffic profile," which is defined as the characteristics of the network traffic generated by that host which makes it uniquely distinguishable from among those traffic generated by a crowd of its peers. This traffic profile can include IP addresses, Ethernet addresses, content-, and statistical-signatures. The first assumption of this thesis (later described in § 3.5) is that a host's traffic profile has a statistically significant, measurable, one-to-one relationship with the host's identity. "Anonymity" is then defined as the absence of identity – the inability to determine the true origin, given a segment of traffic. A host is said to possess anonymity (is "anonymized") if – post-transformation – the profile of that host's modified traffic no longer has a strong one-to-one relationship with

its identity.

Given these definitions for identity and anonymity, and their direct inverse relationship, the duality in the separate problems of measuring anonymity and identity is apparent.

3.2 Threat model

The threat model considered in this thesis is the scenario where an organization such as an academic institution, corporation, or government entity, wishes to release a packet trace to the public, for research purposes, post-intrusion forensics, or other purpose. A full network traffic capture may contain sensitive information including personally identifiable information about users, artifacts which may leak information about the security policy of the network (such as security software and operating system updates), information which reveal the structure of internal network, information that reveal distinct types of user activity (such as the use of torrent software) – to name a few. In this scenario, the releasing party uses existing content-obfuscation tools such as `tcpmkpub` to remove overt signatures such as the payload information and certain header flags, then uses `CryptoPan` to perform a one-way prefix-preserving transformation on the IP addresses. Believing this to be a sufficient anonymization transform, the organization releases the dataset. An attacker then takes this dataset, and through the use of statistical modeling of packet header fields, recovers a list of potential Linux systems running outdated kernel versions which has a TCP-stack implementation that always responds in a predictable manner, such as by setting specific initial window sizes. The IP addresses were obfuscated so the attacker does not know the true addresses of the vulnerable targets. However, he observes a subnet port-scan within the dataset. Given that he knows the dataset was most likely obfuscated using a prefix-preserving IP-address transformation, and given that subnet port-scans will usually scan linearly, starting from 0 and ending in 255, he is able to identify all of the hosts in this subnet after having identified one of them, simply by tracing the port-scan. Scanning the traffic for this subset, he observes incoming web traffic on one host. Since the attacker knows which organization released the data, he connects to the web site of that organization, and collects the IP addresses for their web servers. Now he knows which subnet the vulnerable

machines are located and needs only a few attempts before he is able to find and exploit the vulnerable host.

Instances similar to this scenario have been observed in the past. The technique of passive OS fingerprinting is well known, and the popular tool known as `p0f` [Zalewski, 2006] serves exactly this purpose – identifying operating systems and versions using statistical features of the TCP SYN packets. Discovering subnets based on port-scan artifacts within the dataset has similarly been discussed by previous authors [Fan *et al.*, 2004]. Further, actual cases of researchers using statistical models to de-anonymize publicly released datasets do exist. One example of such a case would the work by Coull *et al.* [Coull *et al.*, 2008] where the authors attempted to reconstruct the layer-2 infrastructure of the LBNL public research dataset.

In addition to role-based host discovery. User-privacy issues are also of concern. This may include instances of use of prohibited software such as Bittorrent. The data provider may wish to obscure the number of hosts using such software on the network in order to prevent the identification of specific hosts. Past work has demonstrated the feasibility of inferring web-sites based on the statistical profiles of connections within anonymized Netflow logs [Coull *et al.*, 2007a]; in this case, anonymity by merging traffic among a crowd of similar peers, would be needed to provide further anonymity to the source hosts.

Similar to role-based discovery and user-privacy, vulnerability-disclosure is a potential type of information leakage. Typically, when a host is successfully compromised by a non-directed, indiscriminate, attack, it is often turned into a bot or other pivot platform for further attacks against other hosts. This post-exploit behavior is often distinguishable, especially in the case of botnets [Kolesnikov and Lee, 2006; Fogla *et al.*, 2006]. Vulnerable hosts, such as those running older and outdated versions of common operating systems, may also leak signatures such as those used in passive-OS fingerprinting. The distribution of such hosts on a network is a feature that the data provider may wish to obscure. Security policy-actuation profiles such as automated anti-virus update behavior and internal firewall policy, network segmentation, DNS and other systems architectures, may be considered private information and not to be released by the provider. These problems are all within the domain of mapping behavior profiles to host identities, and the obfuscation of this

mapping is one avenue for the anonymization of the system as a whole.

A naive way to prevent such attacks is to simply re-map all IP addresses within a network to a single pseudo-IP address. Thus, all activity is essentially re-mapped to one single host with very large volume of traffic. This, however, is an obviously unattractive solution, as it removes any utility that the dataset may have possessed by destroying all statistical distributions within the dataset. It is essential to keep in mind that the released dataset must serve a purpose – distinct behavioral characteristics and traffic profiles must be preserved to some extent. The underlying motivation of this thesis is that there is a natural continuum between these extremes of preserving multiple distinct hosts which features high utility and low anonymity, and a single distinct host which features high anonymity but low utility. The exploration of this continuum using machine learning-based methods for traffic behavior is the core of this thesis.

3.3 Scope of this thesis

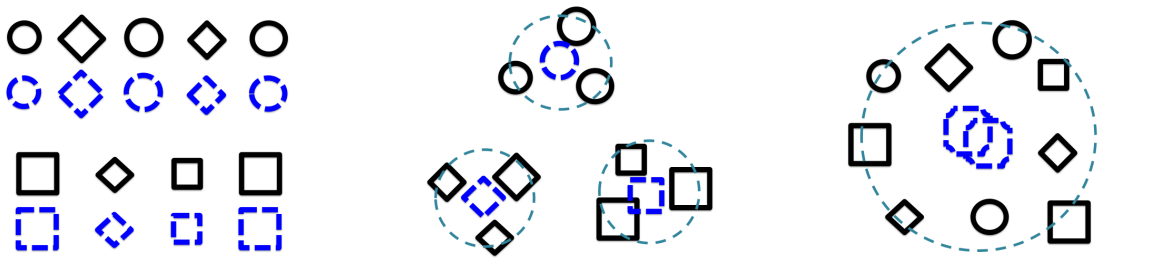
This section describes the data studied in this thesis and scope of this work. We provide a brief description of packet-trace data and its organization, followed by a description of our feature set and the justification for using this set. As stated previously, this thesis focuses on the host-layer behavior profiling. For each host, a single behavior model is extracted based on three main features: the types of services used, the volume of information per service, and the transition between services. Specifically, only traffic that is transferred over TCP is used for measurement. TCP is a full-duplex, session-oriented, protocol that is used when a reliable medium for information-transfer over the Internet is required. Given that it is the primary information exchange channel, it is more strongly correlated with user intent and behavior than other protocols such as ICMP or UDP.

By focusing on service, volume, and transition, it is possible to extract behavior profiles from both packet-trace as well as Netflow data. The service information is obtained simply from the destination port of the TCP session. The volume information is measured as the number of packets transferred per session. The transition information is tracked simply by measuring the change in these values over time. This versatility permits comparisons

of different data in different but relatable formats across different source providers. Exact details of feature extraction is discussed in a later section.

Content is not considered for the feature set for several reasons. Primarily, modeling content is an entirely different problem than modeling network behavior, and is, for the most part, beyond the scope of this thesis. Content is diverse across protocols, and for many services, represent dynamic user-generated information which may be incongruent with statistical measurements of network behavior. In addition, given the scale of enterprise-level networks and ever increasing bandwidth (greater than 10GBps in most enterprise environments as of the year 2012), most traffic monitoring systems do not capture full packet traces for extended periods of time; instead, they capture packets headers only, or rely on Netflow captures. This means that content is not always available for use in measurements. Finally, the potential for side-channel leakage is is naturally higher when using content, and quantifying risk becomes significantly more challenging, beyond this scope of this study.

Finally, we make the distinction between anonymization framework and policy. Our system is meant to be a framework with which we can extend both our understanding of packet-trace anonymization and the current state-of-the-art technologies, such as those found in `tcpmkpub` and `PktAnon`. Our methods are meant to be congruent with these prior methods and not a substitute. For these reasons, we do not need to be concerned with certain technical challenges such as prefix-preserving IP-address transformations, payload hashing, and certain header-value obfuscations. It is assumed that these methods are available and can be applied to the results of our anonymization procedure. Further, the distinction between anonymization system and anonymization policy must be made. Unlike microdata anonymization, where data is fully encapsulated as numbers within a table and measurements of anonymity can be well-bounded and exact, where anonymization translates to altering entries within a table in order to shift metrics such as the means or standard deviations beyond certain thresholds, network trace anonymization for Netflow and PCAP data is a different problem – one where such exact measurements for anonymity are not available. Such data is inherently multifaceted and side-channel attacks against anonymization platforms have not been fully explored; new attacks might remain unknown



(a) Per-host obfuscation (low) (b) Proposed Solution (c) Randomized mixnet (high)

Figure 3.1: Anonymization paradigms are shown: original traces shown in solid-black, new traces shown in dashed-blue.

until it is exploited. Network-trace anonymization, much like the rest of the security field – as stated in prior related literature – is a study in risk minimization.

In this thesis, we provide the motivation for our approach, explain the derivation of our system, demonstrate quantifiable effectiveness in our experiments, and show how our model relates to existing measurements for anonymity, as well as how to measure this relationship.

3.4 Behavior-based traffic-mixture for anonymization

Consider the diagram presented in figure (3.1). In this figure, three different shapes (triangle, circle, diamond) are used to conceptually represent different types of behavior patterns. In panel (a) the anonymization paradigm of per-host obfuscation is shown; this results from using tools such as `tcpmkpub` to sanitize each host independently from one another. Such transforms preserves the most amount of utility, from a measurement standpoint, in that the statistical properties of the dataset is untouched for the most part. However, it yields the least amount of anonymity as these statistical properties may encapsulate fingerprints which are just as discriminative as the content-signatures that were removed. Figure (3.1)(c) represents the other extreme, where the network traffic is sent through a randomized mixnet such as Tor, making all traffic appears to originate from a single host. This provides the highest degree of anonymity, while at the same time destroying all of the useful statistical information by aggregating all statistical behaviors into a single average. (b) represents our proposed solution. In this approach, we use machine learning to identify groups of similarly-

behaving hosts, then merge the traffic among these hosts and assign new group-identities to the resulting set of independent crowds. Anonymity for any particular host is obtained from being having all of that host's traffic mixed among traffic belonging to similarly-behaving peers, where re-attribution – as a signal-separation problem – is measurably difficult. Also, as an intrinsic property, the dilution of the aggregate statistical distribution of among such crowds is minimized.

3.5 Fundamental assumptions

The following assumptions about the characteristics of the network traffic form the foundation of our methodology.

Assumption 1: Network behavior is, in general, idiosyncratic
The behavior of a host on a network, whether driven by interactive human operators or automated service daemons, is – for the most part – idiosyncratic and predictable, given sufficient data.

While network devices may be similarly configure with the same operating system, application set, *etc*, this assumption states that no two devices on a network will utilize these applications in exactly the same way such that the generated traffic from these machines, with respect to diversity, volume, rate, and other features, are exactly alike and indistinguishable from one another. In general, unique behavior is recognizable given accurate behavior models and sufficient training data.

Assumption 2: Behavior can be modeled using meta-data such as packet headers
Host behaviors can be quantified and modeled using machine-learning-based time-series statistical models, using only captured packet headers, without payload content.

Fundamentally, behavior is not conditioned on the content of packet payloads – or more

generally, the content of the traffic streams. While behavior – in terms of measurement of network statistics – may be a side effect of information transport, and though there may be a strong correlation between the content of the traffic stream and the statistical profile of the traffic, such profile can be encapsulated by models that are trained while remaining agnostic to content. This is a characteristic that all behavior-based systems – especially anonymization systems – must seek to achieve, given that data providers rarely offer un-obfuscated packet payload content.

Assumption 3: Group similarity is identifiable

Given a sufficiently large network, groups of similarly behaving hosts will emerge, and can be identified using these statistical models. These hosts can be reassigned to represent single identity with minimal information dilution between the original individual behavioral statistics and the group aggregate statistics.

The core of this thesis is that anonymity-by-crowds can be achieved where all members of each crowd exhibit measurably-similar behavior. The behavior-crowd serves to anonymize each member and to preserve the statistical profile that characterize these member in the aggregate traffic mixture. This, of course, implies that crowds of similar behavior exist, and can be identified using statistical measurements.

Assumption 4: Traffic can be mixed such that re-matching is difficult

Absent the inclusion of side-channel information beyond the scope of behavior modeling – such as content or other direct labels –, traffic can be anonymized with the group identities such that re-attribution back to particular individual hosts within that group is quantifiably difficult.

"Quantifiable" implies that it is possible to derive a method with which we can measure whether one anonymized-mixture is better than another. Further, that we can measure certain related qualities such as minimum mixture-size and ensure that these requirements are met.

Assumption 5: Traffic with targeted behavior can be synthesized
Given a target behavior model, it is possible to synthesis packet and flow-layer traffic that is consistent with this model.

This assumption states that a generative model for behavior can be constructed, and that it would be possible to sample new instances of network traffic given this model, and that these traffic instances would be measurably similar to the original.

3.6 Privacy and utility

In our approach, privacy depends on several measures. Primarily, these include the *size* of the mixture and the *homogeneity* in the behavioral characteristics of the members within that mixture. In the former, the size of the mixture relates directly to the well-understood concept of anonymity-set-size – the k value in k-anonymity. All factors equal, the size of the anonymity set is a direct measure of the level of privacy provided to each member. This is directly related to the re-attribution problem: given a sample of traffic, without side channel information, the probability of accurate prediction is, in general, $1/k$. In the latter measure, the homogeneity in the cluster behavior is directly related to how similar each member is to every other member, and is directly measurable as the sum of the pair-wise similarities under some behavior model. If a web-server, which observes thousands of connections per minute where each connection is the handling of the same static HTTP request, is mixed with a crowd of user machines where no other member is receiving port 80 traffic, and all members are observing fairly high-latency intermittent connections, then the web-server would immediately stand out. This is a measurable loss in anonymity.

First and foremost, this thesis provides a machine-learning-motived framework by which we can measure network traffic behavior with quantifiable accuracy, and segment this traffic distribution as the solution of a graph-partitioning problem where maximization of cluster-homogeneity is an intrinsic property of the solution. We further show how to estimate a minimum required anonymity set size (*i.e.* cluster size) and how to guarantee that this

requirement is met.

The properties which we hide and preserve is enumerated here in this section. As previously mentioned, anonymization is a matter of policy and risk minimization – where attacks often consists of side-channel information. However, certain invariant properties exist for all anonymization systems. For example, network addresses such as IP addresses and MAC addresses must be hidden. Content of packet payloads must also be hidden, as this data would offer a potent channel for information-leakage – it doesn't matter well the anonymization system is able to mixture a user's traffic, if the packet payload exposed the username that user typed into a web form. A range of passive network TCP-implementation-centric features, such as elements within packet headers, and how these features should be obfuscated are discussed in the later chapter on anonymization.

Related to this, certain properties of the data must be preserved in order to maintain the utility of the dataset. These include but are not limited to:

- Volume distribution - the distribution of traffic volume in terms of total number of packets, flows, and volume of data transferred.
- Rate/frequency distribution - the distribution on traffic frequencies such as inter-arrival times and transmission rates.
- Service/protocol distribution - this is information pertaining to the range of services on the network, measurable as a function of uniqueTCP port values.
- Distribution of host behavior - the distribution of distinct behavior types on the network such as servers, user machines, etc, should be preserved to the best extent.
- Network characteristics - feature such as path MTU, TCP sequence numbers, and related values. Naturally, some values must be changed in oder to provide mixture-based source anonymization, however the distortion should be minimal.
- Communication structure - the anonymization transformation should not destroy the topology information of the underlying network.

In this thesis, we should how these properties for privacy and utility are maintained.

3.7 Evaluation methodology

The methodology proposed in this thesis is based primarily on behavior modeling, graph-partition-based clustering, and traffic synthesis. Fundamentally, this is an unsupervised learning problem where we do not have true labels for which hosts should be mixed together. However, there are natural ways to quantify the accuracy of the proposed solution. Recall that our measure of anonymity is conditioned on the cluster size, and the homogeneity within that cluster. This implies that two important measurements are necessary. The first is the accuracy of the behavior model, and the second is the accuracy with which we measure similarity between these models. And finally, we need to measure the fidelity of the synthesized data. These methods are evaluated as follows:

- **Behavior model**: We evaluate the accuracy of the model with classification-evaluation. Since no true labels are available which hosts should be matched, we evaluate accuracy by segmenting a host's own traffic and matching that traffic with itself. Models are learned on the training data and evaluated using the testing data. Under this condition, the problem is transformed into a supervised-learning set-up. Classification is based on select the host model that produces the highest likelihood on the test data.

- **Similarity metric**: We derive a kernel-based similarity metric based on the behavior models. Evaluation is done by using this kernel in classification evaluation and visual evaluation using embedding.

- **Synthesis**: Measuring the fidelity of the synthetic data is done by evaluating the likelihood of the data using the models that they were synthesized to represent. Comparisons with authentic and randomly-sampled data confirms the fidelity of this data. Visual confirmation using embedding is also used.

Measurements for the statistics preserving aspect of this transformation are also provided:

- **Volume distribution**: Volume is a feature in our behavior model, and the anonymization transformation preserves the shape of the distribution. This is demonstrated by compare the distributions before and after transformation.

- **Port distribution**: Port values and their transitions are features within our model, and the clustering algorithms naturally preserve this distribution. This is demonstrated in the experiments.
- **Behavior distribution**: Preservation of the behavior distribution is an intrinsic property of our clustering algorithm. This is shown in the experiments.
- **Anomaly Detection**: Congruent to the above property, anomalies in the original traffic translate to anomalous clusters in the resulting data; while the cluster itself provides anonymity against re-attribution.
- **Network-centric characteristics**: The nature of our anonymization transform applies minimal obfuscation to these features, such as path MTU, TCP sequence numbers *etc.*
- **Topology**: While not explicitly modeled in our framework, we show how we can naturally extend our models to make use of topological information about the network.

In addition, we show how our models related directly to well-known measurements in the field of anonymization. These include k-anonymity, l-diversity, t-closeness, and anonymity-set-size. So that theories that make use of these measurement would be applicable to our framework.

Definitions and notations

All mathematical variables in bold font such as $\mathbf{x}$ and $\mathbf{y}$ denote column vectors. Matrices are denoted using bold capital letters, such as $\mathbf{A}$. Non-bold variables, such as d, denote scalar values. We use $\mathbf{x}_i$ to denote the i^{th} vector of a set of vectors $\mathbf{X} = \{\mathbf{x}_1, .., \mathbf{x}_N\}$. The $(i, j)^{th}$ component of a matrix is denoted $\mathbf{x}[i, j]$. $\mathbf{x}^\top$ denotes the transpose of $\mathbf{x}$. Other notations that are not mentioned here will be described withn the sections that they are used.

Chapter 4

Statistical Model for Network Behavior

Introduction

Behavior modeling is used in many areas of network research, and span a broad range of topics. These range from host-similarity measurements in communication-structures [Iliofotou *et al.*, 2007; Tan *et al.*, 2003] to anonymized protocol prediction [Wright *et al.*, 2006] and web traffic prediction [Coull *et al.*, 2007a], to ubiquitous use in statistical anomaly-detection [Wang *et al.*, 2005; Fogla and Lee, 2006; Wang *et al.*, 2006; Song *et al.*, 2009]. This is in addition to measurement research such as route optimization, and load balancing, to name a few. Our work is distinguishable from these others in that, rather than focusing on a specific network measurement, our models are designed to capture a high-level profile for host behavior – one that incorporates traffic from all exercised protocols and services, as well as information relating to how these features relate to each other change over time. Our goal is not to produce the most accurate model for any specific protocol, but rather one that is sufficiently accurate to summarize the overall behavior characteristics of a network host and quickly optimized over a large quantity of training data; importantly, it should support a valid Euclidean similarity metric such that differences in behavior can be measured.

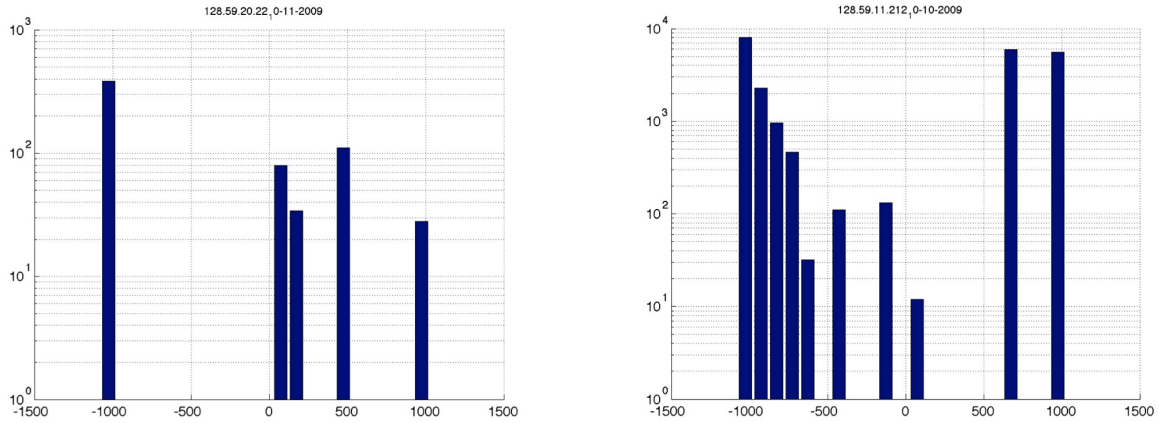
This goal is realized in a model which we refer to as the protocol-transition Markov model (PTM). The PTM a state-based model that tracks the range of network services utilized by a host (HTTP, FTP, *etc*) as function of TCP port values. The volume of traffic sent on these services is measured with exponential-family emissions distributions that are coupled to each state, and the transition among these services is tracked using a transition-probability table. In many aspects, the PTM is similar to other state-based models in machine learning such as the hidden Markov model. However, as we will demonstrate in the result comparisons, the assumption of the existence of a set of latent states – representing abstract subclasses of behavior – which work well in other feature domains, can be inefficient in high-level network-traffic modeling. We show that superior performance is achieved by simply coupling the states with the services directly. Another benefit of this approach is that optimizing the model parameters for the state transitions is much simpler and can be achieved optimally using maximum-likelihood estimation that grows with time linearly proportion to the sample size.

This chapter discusses the features used, the models, and derivations and our motivations

for these choices. We compare the performance of the PTM with similar models in this category and demonstrate the superior performance of this approach. The reader is assumed to be knowledgeable in networking fundamentals as well as statistics and machine-learning theory.

4.1 Feature extraction

The first challenge in behavior modeling is selecting the feature set with which to model. The features must summarize the characteristics of the traffic while, at the same time, be discriminative enough in order for differences in behavior among hosts to stand out. The choice of feature set also depends on the computation cost of of the training algorithm. Models with assumptions on hidden states, for example, cannot be optimized using maximum likelihood (ML), thus multiple passes over the data is necessitated using stochastic gradient-descent algorithms such as expectation-maximization (EM). In the context of network traffic, the dataset often spans several hundred gigabytes of information. The Columbia University network traffic dataset, for example, contains three weeks worth of traffic and spans over 7 billion packets. Multiple low-level passes over such large-scale data is computationally infeasible. Realistic assumptions on data-availability should also be followed: it is rare that providers would include packet-payload content in traffic datasets. This is more often to due privacy concerns as well as computational constraints. As an example, full packet captures on Columbia University's Computer Science department's network yields well over 2Gb worth of traffic every minute – consistently. This volume is typical of modern gigabit-speed enterprise networks. Quantifying behavior depends on time, and capturing such data at such scale over prolonged periods is often too demanding. Most datasets, therefore, provide only statistical records, such as Cisco Netflow captures or packet-header captures only. This is shown in the Department of Homeland Security's Predict.org [PREDICT, 2011] data-sharing center, where large network traffic datasets are shared amongst different research organizations. For this and similar reasons, network research often focuses on more abstract statistical properties; volumetric information in particular. Many prior works has demonstrated the accuracy of using traffic volume, and packet/flow-size information.



(a) User machine port-histogram

(b) Server port-histogram

Figure 4.1: Log-scale plot of histograms for traffic-volume by ports. $x < 0$ range represents Received-On ports, $x \geq 0$ represents Sent-To ports. Notice the distinct behavioral patterns between the user and server machines.

As a simple introduction, this section and the following, discusses our fundamental approach towards volume-based behavior modeling. The feature representation is discussed and the exponential-family class of statistical distributions is described, and the extension into time-series modeling with hidden Markov models is covered. The limitations of this approach is highlighted and used to lead into our main work, which is the derivation of a more efficient time-series model that utilizes the same sort of features.

Our results show that network-behavior can be accurately modeled by observing the amount of traffic sent to and received on each port. TCP ports are, in general, strongly correlated with the type of service exercised by that host (SSH traffic uses port 22; HTTP use 80). By default, the selection of TCP ports are determined by the operating system, and not by the application or user, according to TCP specifications for well-known and reserved ports, making TCP ports a consistent feature across different hosts. This is true for the most part – applications and operating systems often allow the user to select different ports, however, in the majority of cases, TCP port values is a strong indication of the type of service being used. For example, outbound requests sent to port 80 indicate a request for web-traffic; port 25 indicates mail traffic; port 22 indicates encrypted sessions such as

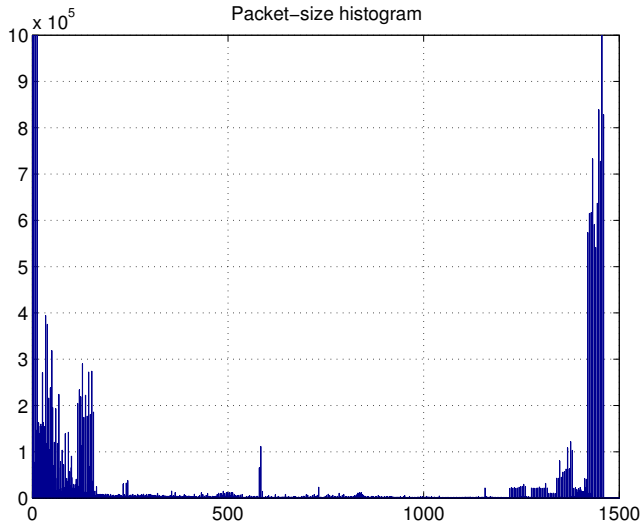


Figure 4.2: Distribution of packet sizes during two hours worth of traffic, randomly sampled over a period of 20 days.

SSH, and so on. Conversely, if a host receives significant traffic on port 80, it is a strong indication that it is a web-server. The volume of traffic observed further distinguishes one host from another.

Monitoring service-port ranges provides a view of the types of interactions a host engages in within that time frame, and given sufficient sampling, distinct fingerprints of behavior emerges. Figure (4.1) 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 the user machine make 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.

Considering the fact that TCP represents a state-full session layer, different classes of packets exist. These are packets that transfer content and those that manipulate the state of the session, such as SYN, ACK, FIN packets. When modeling behavior based

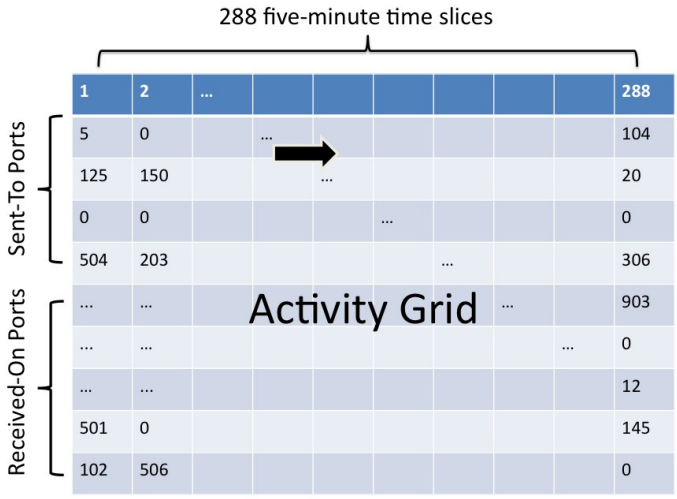


Figure 4.3: Activity-grid representation: 288 five-minute port histograms; a time-series feature representation for network traffic.

on packet-counts alone, the distribution of packet types is ignored. This is a potential oversight given that packet sizes have been shown to be a discriminative factor in protocol classification [Wright *et al.*, 2006]. However, with respect to high-level behavior modeling at the service-distribution layer – which is the scope of our study – the level of detail at this view does not require the use of packet sizes. Figure (4.2) demonstrates this point. The distribution of packet sizes is plotted for our network, sampled over a period of two hours at random intervals over the course of 20 days (our network uses Ethernet, *i.e.* 1500-byte MTU.) When agnostic to specific protocols (HTTP, FTP,...*etc*), the overall distribution of packets sizes is clearly a bimodal distribution representing the two distinct classes previously mentioned. Overall, the true volume of data transferred is related to the packet count by a multiplicative factor.

Given that host behavior is changing process, a holistic view of traffic at the granularity of a single day, such as the one shown in figure (4.1), cannot capture a representation of this change. A time-varying view is necessary – one that shows how these activities change across the span of hours, days, and possibly weeks. The straightforward extension of this representation is to capture the histogram on smaller intervals and use a collection of such

intervals as the feature set. Figure (4.3) shows the time-varying activity grid representation. This structure is 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 in the format of Hour:Minute:Second. Using 2050 features to represent input and output activity gives us a 2050×288 matrix. Similar feature sets are discussed in [Coull *et al.*, 2011]. The columns within the grid represent snapshots of behavior across time. The next section discusses time-series models that can fit to this type of data. This form of representation is often very sparse, and model estimation can be an ill-posed problem. The following subsection discusses how to compress this data using a sub-space projection-based method.

4.1.1 Dimensionality reduction with sub-space projection

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, generalization performance becomes a problem, as it is known that as the dimensionality of the training data increases, so does the requirement for more instances of such data, to avoid under-fitting the models. The second problem is the memory requirement – using 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 hundreds of thousands of hosts, and in our tests using a subset of Columbia's traffic with roughly 7,000 hosts, full-memory consumption and disk-thrashing occurs roughly 10 seconds into the processing of the algorithm in this simple setting. To solve this problem, dimensionality-reduction algorithms can be used to compress the data into a smaller subspace where under-fitting is less of an issue, and memory requirements are much more relaxed. The methods discussed in this section consist of subspace projection-based algorithms. This approach consists of learning a projection matrix $\mathbf{P}$ such that the new samples are derived by the following linear relation:

$$\mathbf{Y} = \mathbf{P}^T (\mathbf{x} - \boldsymbol{\mu}), \qquad \boldsymbol{\mu} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_i.$$

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. For Principal Component Analysis (PCA), the

columns of $\mathbf{P}$ consists of the top eigenvectors of the covariance matrix $\mathbf{\Sigma}$, recovered from the samples:

$$\mathbf{\Sigma} = \frac{1}{N} \sum_{i=1}^{N} (\mathbf{x}_i - \boldsymbol{\mu}) (\mathbf{x}_i - \boldsymbol{\mu})^T .$$

Further, the Nyström method for low-rank approximations [Zhang *et al.*, 2008] can be used to obtain an estimate for the covariance matrix at significantly less computational costs with some sacrifice of precision. Kernel-based dimensionality reduction is discussed in a subsequent section. We propose to further study this and other dimensionality reduction techniques, including but not limited to, discriminant analysis-based projection methods such as Fisher Linear Discriminant which maximizes the inter-cluster separation instead of maximizing the covariance in the subspace, as in the PCA case.

A significantly more computationally-efficient approach – one that is more suitable for large network-scale datasets – is to use Random Projection (RP), where $\mathbf{P}$ is randomly initialized and then made orthonormal via the Gram-Schmidt process. In practice, random projection is a much faster and simpler approach that yields results comparable with PCA. Given the type of distribution, bounds on the performance of RP in relation to PCA are available, these are a function of the difference in the eigenvalues of the covariance matrix [Dasgupta, 2000].

4.2 Volume distributions modeling

Time-series models are typically composed of two parts: a state representation and transition model and an independent emissions model that is coupled with each state. This feature is present in discrete latent-space system such as hidden Markov models [Rabiner, 1989] which uses discrete sets of hidden states which must be summed over when performing inference, as well as continuous systems such as the Linear Dynamical System [Ghahramani and Hinton, 1996] that uses a continuous state model. Fundamentally, time-series models assume the data is generated from a closed system with multiple states, that the emissions model coupled with any given state is responsible for the observed output data. Shifts in output patterns are attributed to transitions of (potentially latent) internal states. For vol-

umetric distribution modeling, we evaluate the performance of several related distributions that represent fundamentally different assumptions on how the data is generated. These distributions are closely related methods, drawn from the exponential-family of distributions. Evaluation of these methods provides insight into how to best interpret network traffic features, and are used as motivation to derive the PTM in a later section.

Given that counting features are used, the multinomial distribution is a natural choice. The multivariate multinomial distribution calculates the probability of each sample based on the ratio of the vector components.

$$p(\mathbf{x}|\theta) = \frac{n!}{\mathbf{x}(1)!, ..., \mathbf{x}(k)!}\theta(1)^{\mathbf{x}(1)} \cdots \theta(k)^{\mathbf{x}(k)}, \qquad \sum_{i=1}^{k} \mathbf{x}(i) = n \tag{4.1}$$

The θ variable constitutes the sufficient statistics of this model and the normalized ratios of events $\mathbf{x}(i), ..., \mathbf{x}(k)$ in the training data. Multinomial distributions model discrete event occurrences as separate independent events, and proportion of these occurrences in relation to one-another. This entails fingerprinting a host's network behavior based on the ratio of services invoked, at each time step, as opposed of absolute volume. This allows hosts with similar distributions in service-use but dissimilar in volume, such the difference between as a small web server a large one, to appear similar to one another independent of exact volume.

The second most relevant model is the multivariate Gaussian distribution, also known as the multivariate-normal (MVN) distribution. This model is used when we observe the emission as a single independent-identically-distributed (i.i.d.) multivariate variable, and is useful when absolute volumes are tracked as a model of behavior – where the volume-of-traffic and not the ratio-of-services is considered discriminative. The MVN model in this case, treats the observation at each time-step as a single draw from a joint distribution. The MVN is also a continuous model, which allows more flexibility in working with feature set transformations such as sub-space projects and other preprocessing functions.

$$p(x|\mu, \Sigma) = \frac{1}{(2\pi)^{d/2}\sqrt{|\Sigma|}} \exp\left\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right\} \tag{4.2}$$

The third, and simplest, model is the Discrete Conditional Probability Table (DCPT). This model is used more in computational biology and natural language processing, where

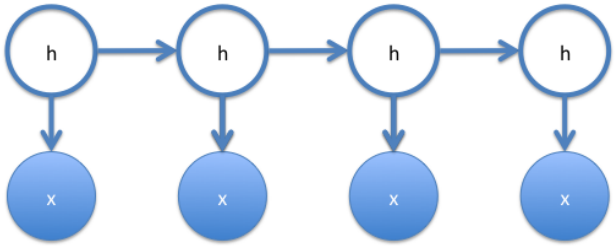


Figure 4.4: Graphical representation of the hidden Markov model. Shaded nodes represent observable emissions. Unshaded nodes represent hidden states.

each emissions can be considered as a discrete symbol from a closed alphabet. In this case, both the emissions model and the transition model are represented by probability tables.

$$p(a|b) = \mathbf{T}(a, b), \qquad \sum_j \mathbf{T}(i, j) = 1 \quad \forall i \tag{4.3}$$

The DCPT approach was studied in the earliest implementations of the hidden Markov model. It is most appropriate when feature sets can be represented as discrete events. With volumetric information, some pre-processing is necessary in order to transform packet counts into discrete symbols. In out experiments, we clustered the range of volumes and assigned unique symbols to each entry based on the cluster label. In general, the DCPT method can be used where a separate system assigns unique labels to discrete events in network traffic, the appearance of these events can then be modeled in a time-series.

4.3 Time-series extension with hidden Markov models

Given the emissions models described in the previous section, the hidden Markov model (HMM) involves modeling the transitions between a set of hidden states, where each state is coupled with an independent emissions model. An HMM is a Markov-transition model where the process transitions between states in an unobservable path; where each state is conditionally dependent on the previous state.

Figure (4.4) shows the graphical representation for an HMM's dependency structure. The un-shaded nodes represent unobserved hidden states. A Markov-dependency is used to model the transition between each subsequent state. The shaded node at each state represents the emissions model. This is a model described in the previous section, such as one from the exponential family of distributions.

Consider an HMM with multi-variate Gaussian emissions ($\mathbf{x}$). The probability of observing a sequence of such emissions is denoted as $p(\mathbf{X}|\theta)$, where $\boldsymbol{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_T\}$ presents the a sequence of observations 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 unimodal Gaussian models ($q_t = \{1, \ldots, M\}$). This HMM is specified by the parameters: $\theta = \{\boldsymbol{\pi}, \boldsymbol{\alpha}, \boldsymbol{\mu}, \boldsymbol{\Sigma}\}$. These variables represent the:

- Initial-state probability distribution $\pi_i = p(q_0 = i)$, $i = 1 \ldots M$
- The state transition probability matrix $\boldsymbol{\alpha} \in \Re^{M \times M}$ where $\boldsymbol{\alpha}_{ij} = p(q_t = j | q_{t-1} = i)$
- The Gaussian emission model $p(x_t|q_t = i) = \mathcal{N}(\mathbf{x}_t|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$, 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 parameters of the Gaussian model 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. The scalar variable d is the dimensionality of $\mathbf{x}$. Since the exact value of q_t is unknown it is referred to as a hidden state. The transition through a sequence of hidden states $\boldsymbol{q} = \{q_1, \ldots, q_T\}$ and the emissions at each step determines the generation of the observed sequence $\boldsymbol{X}$. Further, because the exact hidden states are unknown, $p(\mathbf{X}|\theta)$ must be evaluated over all possible combinations of the hidden states ($\boldsymbol{q}$), which translates to evaluating the emissions probability at each setting and summing over all possible settings in order find the normalized probability for the given sample $\mathbf{X}$:

$$p(\mathbf{X}|\theta) = \sum_{q_0} \cdots \sum_{q_T} p(\mathbf{x}_0|q_0)p(q_0) \prod_{t=1}^{T} p(\mathbf{x}_t|q_t)p(q_t|q_{t-1}). \tag{4.4}$$

Brute-force evaluation of equation (4.4) would normally require exponential time, however, dynamic-programming-based methods such as the Junction-Tree algorithm (JTA) or

the Forward-backward algorithm, is often used to compute this quantity efficiently.

Estimating the parameter (θ) of an HMM is typically done via expectation maximization (EM). This process includes two iterative steps: the expectation step where the likelihood of the data is calculated using the current estimate of the model parameters and a maximization step where the model parameter is adjusted by moving it in the gradient of the likelihood function over the training data. The EM update rules are conditioned on the emissions models are derived differently for each model. Our derivations for the HMM with multi-variate Gaussian emissions are presented here.

The E-step uses a forward pass to obtain posterior marginals over the hidden states given the observations:

$$\gamma_t(i) = p(q_t = s_i|\mathbf{x}_n, \theta) \tag{4.5}$$

$$\xi_t(i,j) = p(q_t = s_i, q_{t+1} = s_j|\mathbf{x}_n, \theta). \tag{4.6}$$

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:

$$\hat{\pi}_i = \gamma_1(i) \tag{4.7}$$

$$\hat{\alpha}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1}\sum_{j=1}^{M} \xi_t(i,j)} \tag{4.8}$$

$$\hat{\boldsymbol{\mu}}_i = \frac{\sum_{t=1}^{T} \gamma_t(i)\mathbf{x}_t}{\sum_{t=1}^{T} \gamma_t(i)} \tag{4.9}$$

$$\hat{\boldsymbol{\Sigma}}_i = \frac{\sum_{t=1}^{T} \gamma_t(i)(\mathbf{x}_t - \boldsymbol{\mu}_i)(\mathbf{x}_t - \boldsymbol{\mu}_i)^\top}{\sum_{t=1}^{T} \gamma_t(i)}. \tag{4.10}$$

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 update rules for HMMs with Multinomial, DCPT, emissions are derived in the same way; these derivations are simple and (given that the MVN model is the only one relevant in further sections) are omitted for brevity.

4.3.1 Inferring properties about network data from HMM performance

The purpose of this section is to show how the performance disparity between these models infers certain properties of network data and their effect on these models. For testing, we collected three weeks of worth of packet-header-only traffic from Columbia University's computer science department's network using a SPAN port; this dataset is referred to in this thesis as the CUCS dataset. Over the course of three weeks in 2010, over 1.6 millions hosts were observed, generating over 7 billion packets and spanning over a terra-byte of data. Much processing was needed in order to extra useful information for this dataset. First, a subnet of Columbia hosts must be isolated – otherwise the top hosts in the dataset would be foreign hosts such as google.com. In addition, the packet-count distribution is heavily skewed towards a small set of hosts that generate the majority of traffic – a property of most networks. After isolating the traffic of a specific Columbia class C network, and taking the top hosts with sufficient amount of traffic (details discussed in a later subsection), we are left with 80 Columbia hosts that generate the majority of the traffic observed on the network. These data processing steps are described in more detail in § 4.6.1.

These models evaluated by measuring the accuracy of using these models to classify traffic belonging to a particular host (unseen during training) back to that host during testing. The clustering algorithms presented later in this thesis which make up the foundation of the behavior-based anonymity-by-crowds algorithm, is determined by the accuracy of these models, as accurate models indicate accurate representations of behavior, and thus accurate measurements of similarity. The classification experiment is set up as a one-vs.-all matching problem where each test sample is matched against the model of every host. The label is assigned based on the model that yields the highest likelihood value. The baseline performance for this experiment, using random guessing, would yield 1.25% accuracy. In our experiments, 80% of the traffic for a host is randomly chosen as the training set, and 20% as the test set. Classification is performed at the day-granularity, *i.e.* roughly two weeks worth of traffic are used for training and the third week is used for testing. The reported results are each averages of *five* independent runs. (Note: Subsequent experiments in later sections are done on segments of traffic at session-level granularity.)

Table (4.4) shows the results from modeling network traffic as slices port activity across

Data representation	Baseline	G-HMM	M-HMM	D-HMM	G-Vec.
CUCS Raw	1.25%	—	22%	25%	82%
CUCS PCA	1.25%	10%	—	—	63%
CUCS SESSION	1.25%	28%	22%	25%	80%

Table 4.1: Classification performance of different HMMs on CUCS 80 vs. 80 dataset at the *day*-granularity. Dashed entries indicate infeasible settings. "G-Vec." refers to a Gaussian vector model.

time and tracking the transition of these emissions across time. Each slice of network activity represents statistical measurements of port-traffic volume for a five-minute interval, with 288 such intervals present per day. "CUCS Raw" indicates raw packet counts as features. Gaps of inactivity on the network would be represented with zero-valued entries in these samples. "CUCS PCA" indicates normalized PCA-projected features, and "CUCS Session" indicates PCA-projected features which were pre-processes to be gapless, which translates to explicitly not attempting to model periods of network inactivity. In the table, the "G-HMM" column indicates the performance of an HMM with a multivariate-Gaussian emissions distribution. Likewise, "M-HMM" represents an HMM with a Multinomial vector emissions distribution, and D-HMM represents a scalar emissions model where the output has been translated to discrete symbols; this was done by thresholding the aggregate volume of the observed traffic and assigning different symbols to the different volumes of traffic observed, by order of magnitude. "G-Vec." indicates a single multivariate Gaussian model.

A range hidden state sizes for the HMMs were evaluated and the best results are shown; typically no improvement was noted beyond 5 hidden states. A minimum state count of 2 was used, as an HMM with a single state does not have a state-transition property and is equivalent to the multivariate Gaussian ("G-Vec.") distribution. The results of Table (4.4) can be used to infer several useful properties about network traffic data. The fact that the Gaussian vector model outperformed all of the hidden Markov models is surprising, at first. However, an examination of the distribution of network traffic reveals the reason behind this. Network traffic, unlike other domains where continuous emissions are modeled using time-series data, possesses extreme sparsity. The amount of time a user spends active on a

machine is actually fairly sparse, with periods of sharp bursts of activity. When modeling activity conditioned on time, as in the case of the activity grid, this translates to large numbers of zero-vector emissions. The models, therefore, lock on to a zero-emissions state that contains a high self-transition probability. This essentially means that the HMMs are tracking the behavior for when a host is not active, which induces inaccuracy in the model considering the fact that periods of inactivity is a prominent characteristic if all host behavior.

The sparsity of the data was found to strongly influences model accuracy. When the data is processed to reduce sparsity, through PCA, and a pseudo "sessionization" process, where we simply dropped all of the gaps in the traffic, performance of the HMMs improved. The Gaussian vector case is included here for comparison. While it is useful as a classification technique, the PTM model that we propose in the following section outperforms it in further evaluation. In addition, the PTM model is a proper time-series model and, as such, captures the transitional information in the behavior profile, and allows synthesis of time-series data whereas the Gaussian vector model does not.

4.4 The Protocol-transition Markov model

The experiment results with HMMs have demonstrated that hidden-state models, when trained over sparse traffic, can lead to poor fitting and underperformance. This section discusses what can be learned from these results and how to fix these problems in a new model derivation. A new time-series model and feature representation, which overcomes this sparsity problem and yields better performance overall, is presented in the form of the protocol-transition Markov model (PTM), which operates at a higher-level view of the data, using flow-layer statistics.

4.4.1 From packets to flow-statistics

The HMM experiments have shown that attempting to fit hidden state models on sparse traffic can lead to poor performance. Examining the parameters of the trained models reveal that the hidden states tend to lock onto periods of inactivity as a feature in the behavior

representation – essentially tracking the frequency with which a host is inactive. This is an undesired effect as it diminishes the discriminative aspect of the model: long periods of inactivity is common among all hosts, this similarity was overtaking the characteristics of the actual traffic behavior. When the gaps were removed, the performance improved, though still not achieving the same performance level as a simple Gaussian vector model that tracked the aggregate sum. Additionally, the size of the emissions model compounds the problem; without compression, this model is trained on 2050×1-sized vectors, which means at least 2050 training samples are necessary in order to obtain a full rank estimate of the covariance matrix in the Gaussian distributions. In the activity-grid representation, this translates to over seven-days worth to traffic as a minimum requirement, making underfitting a real possibility. Improvement in performance was noted as sparsity was removed by removing the gaps in the traffic due with PCA and sessionization. Given that network data is often sparse, unaligned and captured only over short periods, a new model that operates at the flow-statistics layer is introduced. This model is trained on session information given in Netflow like format:

$$\mathbf{x} \in \{timestamp, srcaddr, \mathbf{srcport}, dstaddr, \mathbf{dstport}, \mathbf{pktcount}\}.$$

The above representation is a record of a flow connection. This information format is available directly from Netflow. When only packet captures are available, these session-layer statistics can be recovered by two means: the first is replaying the packets into a flow-exporter using tools such as `softflowd` and `nfcapd` which gives you a pcap-to-netflow conversion, and the second – simpler – approach is to use session-extractor tool such as `tcptrace`. Tcptrace reassembles the sessions and returns the statistics in the format above. The network data is then represented as a time-ordered set of these flow-layer records $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$.

The PTM learns a state-transition model using only the src and destination port pairings, along with the packet count feature. The models are trained on a per-host basis, so the source IP address is unnecessary, and our model is agnostic to outbound destinations – tracking the aggregate profile of all outbound behavior, therefore the destination IP address is also unnecessary. Further, the src port distribution is tracked only for the purpose of synthesis. The outbound destination port and the packet count are essentially the sufficient

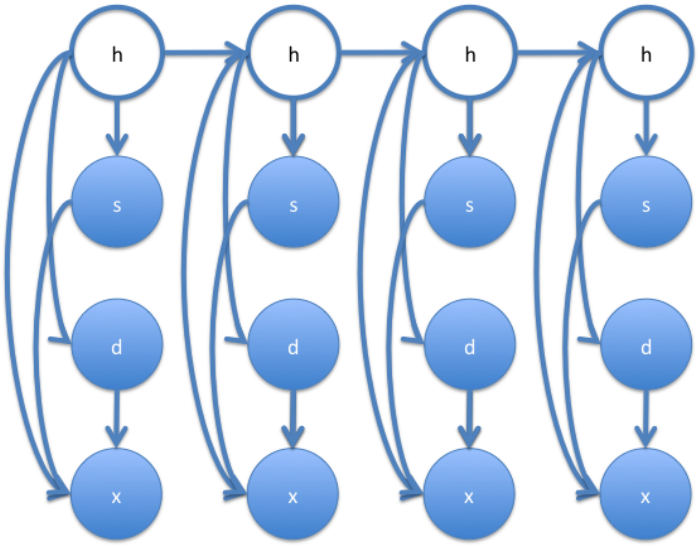


Figure 4.5: Graphical representation of the latent-state flow-transition model.

statistics for this model.

4.4.2 The protocol-transition Markov model

Using this feature set the full latent-state flow-transition model is a hidden-state Markov model with three inter-dependent observed emissions variables: the source port, the destination port, and the flow size. The graphical representation of this model is shown in Figure (4.5). The source and destination pairings identify a unique network application; coupled with the flow size these produce a statistical record which represents a segment of that host's behavior profile. Under this model, a latent variable s represents the current behavior state of the host. The observed output (or "emission") for each state consists of the flow statistics, *i.e.* the source and destination ports, and the flow size, represented by the number of packets observed. Factoring this graphical model yields the following equation. Let $\boldsymbol{x} = \{x_1, x_2, ..., x_T\}$, $\boldsymbol{s} = \{s_1, s_2, ..., s_T\}$, $\boldsymbol{d} = \{d_1, d_2, ..., d_T\}$, and $\boldsymbol{h} = \{h_1, h_2, ..., h_T\}$. Thus we have:

$$p(\mathbf{x}, \mathbf{s}, \mathbf{d}|\mathbf{h}, \mathbf{\Theta}) = \sum_{h_1} \cdots \sum_{h_T} \prod_{t=2}^{T} p(x_t|d_t, s_t, h_t) p(d_t|h_t) p(s_t|h_t) p(h_t|h_{t-1}). \tag{4.11}$$

Where $p(x_t|d_t, s_t, h_t), p(d_t|h_t), p(s_t|h_t)$ are the emissions probabilities and $p(h_t|h_{t-1}$ is the transition probability. $\mathbf{\Theta}$ represents the set of all model parameters for these independent distributions. This full factorization is unnecessarily complex given the nature of network behavior. Using some expert-knowledge of network protocols, it is possible to reduce the model by conditioning both the flow-size and source-port emissions entirely on the destination-port emission. Next we decouple the dependency between the flow-size and the source port. This is reasonable because, given the TCP specification, the destination port represents the type of service requested by that host. Conditioning behavior based on the characteristics of outbound traffic couples the model with the behavior of the type of services requested by the host, and the types of responses sent by the host. Conversely, conditioning the behavior based on input is less accurate as the host does not determine the type of input it receives; rather, that is the behavior profile for the responding host.

Trimming the model in this manner yields the following, more simplified, representation which is factorized based solely on the observed variables, and which allows both easier inference and parameter estimation.

The graphical model, which we call the protocol-transition Markov model (PTM), shown in figure (4.6), takes the following form:

$$p(\mathbf{d}, \mathbf{x}|\mathbf{\Theta}) = p(d_1) p(x_1|d_1) \prod_{t=2}^{T} p(d_t|d_{t-1}) p(x_t|d_t). \tag{4.12}$$

Observe that, in Figure (4.6), that inference does not depend on the source variable. The PTM tracks the dynamics of a system as a transition between specific application services represented by distinguishable protocols (FTP $\rightarrow$ HTTP $\rightarrow$ SMTP, ...). d_t represent the observed destination port of the system at time-step t. Let x_t represents the observed flow-size and is determined by a separate emissions distribution conditioned on the state d_t. The overall interaction is modeled now as a fully observed Markov process. In order to handle the range of behavior types, implicitly assumed to exist given the original latent-space representation, a scalar Gaussian-mixture model (GMM) is used for the flow-size

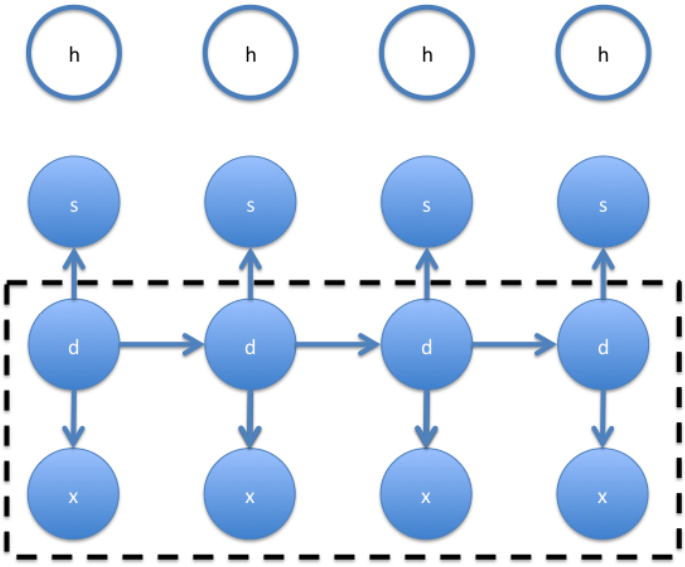


Figure 4.6: Graphical representation of the reduced fully-observed protocol-transition Markov model.

distribution $p(x|d)$:

$$p(x|\theta) = \sum_{i=1}^{M} \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left\{-\frac{(x-\mu_i)^2}{2\sigma_i^2}\right\}. \tag{4.13}$$

Figure (4.7) diagrams this interaction. A mixture model allows us to implicitly capture an estimate of the different types of traffic exchanges observed under a particular protocol. For example, the model-state for port 80 would capture a distribution on the sizes of websites visited; the state for port 25 would track a distribution on the lengths of emails sent, and so forth. The emissions model can be easily estimated, per state, using Expectation Maximization (EM). Since the value of the state is known, parameter estimation for the state-transition probability table $T(a,b) = p(s_t|s_{t-1})$ can be done analytically. Putting these together we obtain the full probability distribution $\boldsymbol{s} = \{s_1, ..., s_T\}, \boldsymbol{x} = \{x_1, ..., x_T\}$ and $\Theta = \{\theta_1, ..., \theta_M\}$:

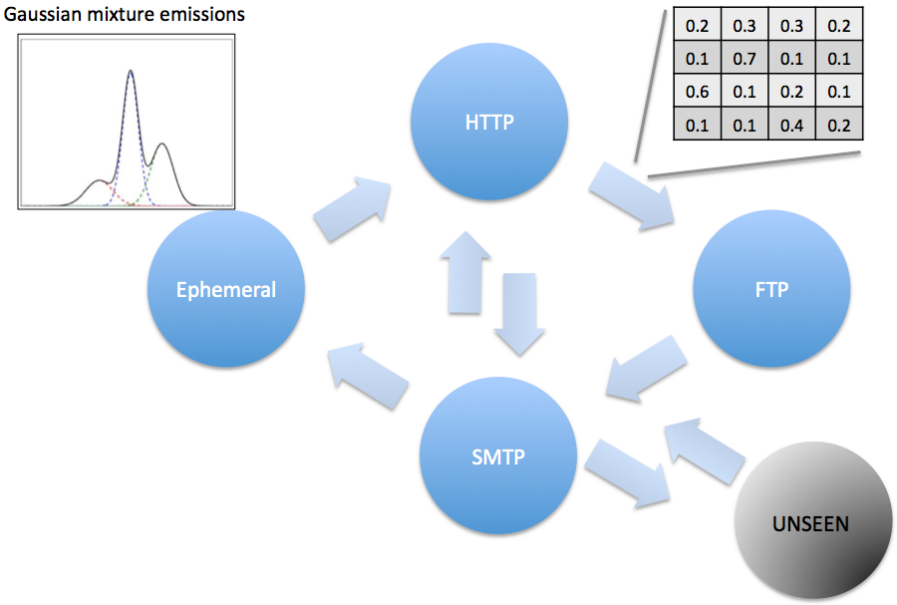


Figure 4.7: Conceptual diagram of the PTM.

$$p(s_1, x_t) = \sum_{t,k} \gamma_t(i,k) \sum_{k=1}^{K} \mathcal{N}(x_1|\theta_{1,k}) \tag{4.14}$$

$$p(\mathbf{s}, \mathbf{x}|\Theta) = p(s_1, x_t) \prod_{t=2}^{T} T(s_{t-1}|s_t) \sum_{i=1}^{M} \mathcal{N}(x_t|\theta_{i,k}). \tag{4.15}$$

For the emissions model of any given state (whose parameter is represented by θ_i), we need to track table of source and destination port pairings. Let $b_{i,j} = p(\text{dst} = j|\text{src} = i)$, this tracks the distribution of source ports which sent traffic to the destination port represented by this state.Table $\mathbf{b}$ is estimated in the same manner as $T(a, b)$. This gives us a final set of parameters that make up the model $\theta = \{\mathbf{b}, \pi_1, \mu_1, \sigma_1, ..., \pi_M, \mu_M, \sigma_M\}$.

4.4.3 Learning the parameters of the PTM

Optimizing the PTM parameters is also via simple expectation-maximization which optimizes the log-likelihood function on $p(\mathbf{X}|\mathbf{\Theta})$ over training dataset $\mathcal{D}$

$$\arg\max_{\Theta} p(\mathbf{X}|\mathbf{\Theta}) = \arg\max_{\Theta} \sum_{\mathcal{D}} \log \left(\prod_{t=2}^{T} T(s_{t-1}|s_t) \sum_{i=1}^{M} \mathcal{N}(x_t|\theta_{i,k}) \right). \tag{4.16}$$

Optimal parameter estimation in the PTM model is dramatically simpler than the HMM case because the states themselves are observed. In the case of the HMM, the hidden state are abstract, and implicitly track sub-classes of similar emissions distributions, therefore their estimation is not subject to maximum likelihood. for the PTM, the states are exactly dependent on the network service, which is identified based on the observable port numbers. This means that optimizing $T(a, b)$, in the ML estimation, consists of simply tracking the port-transition frequencies as follows:

$$T(a,b) = \frac{\#(s_i = b, s_{i-1} = a)}{\#(s_i = b, s_{i-1} = a) + \#(s_i \neq b, s_{i-1} = a)}. \tag{4.17}$$

$\#(s_i = b, s_{i-1} = a)$ in the above equation 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). For each state, learning the mixture-of-Gaussians emissions model is done via standard EM. The E-step uses a forward pass to obtain posterior marginals over the hidden states given the observations:

$$\gamma_t(i) = p(q_t = s_i | \mathbf{x}_n, \theta) \tag{4.18}$$

$$\xi_t(i,j) = p(q_t = s_i, q_{t+1} = s_j | \mathbf{x}_n, \theta). \tag{4.19}$$

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:

$$\hat{\pi}_i = \gamma_1(i) \tag{4.20}$$

$$\hat{\alpha}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \sum_{j=1}^{M} \xi_t(i,j)} \tag{4.21}$$

$$\mu_i = \frac{\sum_{t=1}^{T} \gamma_t(i) x_t}{\sum_{t=1}^{T} \gamma_t(i)} \tag{4.22}$$

$$\hat{\sigma}_i = \frac{\sum_{t=1}^{T} \gamma_t(i)(x_t - \mu_i)(x_t - \mu_i)}{\sum_{t=1}^{T} \gamma_t(i)}. \tag{4.23}$$

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. EM for the PTM is very simple due to the fact that the

emissions is a scalar distribution (representing packet counts.) This means that parameter estimation is both fast and less prone to numerical errors.

4.4.4 Multi-step Markov transitions

The default PTM models the traffic using first-order Markov transitions. This means that each traffic instance is conditionally depending solely on the previous sequence. At the flow-level, and for offline data, this assumption in the dependency structure is not unreasonable, given the level of abstraction found at this layer. Each flow entry capture behavior over a small segments of time and is therefore a smoother representation of behavior than the packet layer; the first-order dependency assumption at the packet-layer, for example, would be much less reasonable. Nevertheless, the extension of this model into a multi-step Markov model is also possible. This can be done by adjusting the dependency based on multiple-time steps as opposed to the previous timestep:

$$p(\mathbf{d}, \mathbf{x}|\mathbf{\Theta}) = p(d_1)p(x_1|d_1) \prod_{t=w+1}^{T} p(d_t|d_{t-1}, d_{t-2}, \ldots, d_{t-w})p(x_t|d_t). \tag{4.24}$$

Here w represents the Markov step size; a step size of 5 would indicate that the probability of transition into the current state is conditionally dependent on the five previous instances. However, this would also increase the size of the transition model exponentially. If a first-order Markov transition table is two dimensions, then a second-order transition model would be geometrically represented as a cube of transition probabilities, and so on. One way to mitigate this is to use the product of the marginals instead $p(d_t|d_{t-1}, d_{t-2}, \ldots, d_{t-w}) \propto p(d_t|d_{t-1})p(d_t|d_{t-2}), ..., p(d_t|d_{t-w})$. This would change the model to the following form:

$$p(\mathbf{d}, \mathbf{x}|\mathbf{\Theta}) = p(d_1)p(x_1|d_1) \prod_{t=w+1}^{T} p(x_t|d_t) \prod_{j=1}^{w} p(d_t|d_{t-j}). \tag{4.25}$$

This would require additional normalization complexity, however if we are only interested in comparing likelihood score for classification or similarity measurements, then the normalization factor can be ignored.

4.5 Technical and implementation details

If we track transition of services based on port values then, in theory, we could have a unmanageable $65,536 \times 65,536$-sized transition matrix. In practice, this is not the case given the design of TCP. There are three types of ports defined in TCP, referred to as the "well known," "registered," and "ephemeral" ports. Each type of port has its own predetermined 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; Bittorrent clients use port 6900, for example. Finally all other services with no registered ports are assigned a randomly selected port from the "ephemeral" range of 49,152 to 65,536. Given this implementation design, two things become apparent. First, 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. Second, since the "ephemeral" range maintains no correlation between the specific service and the port used, we can collapse all activity in this range into a single bin.

Feature extraction for the "registered" 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 the third-party application behavior. Third, it is not a strict requirement for services to use the ports that they are registered to. In our work, we use a histogram to represent this range and bin services based on their numerical proximity. IF we have $49,151 - 1,025 = 48,126$ registered ports, and 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. The results in this paper were derived from models using 20-bin histograms, chosen because port-activity is typically very sparse unless some sort of port-scanning activity is occurring. 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 would induce higher training-data requirements on the machine-learning algorithms 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 $log(x)$. Finally, our model tracks the distribution of port pairings. For each destination port we track an independent set of associated source ports from where we have observed traffic. A problem arises 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 is 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.

4.6 Evaluations

We evaluate the performance of the PTM model over six different network traffic datasets. Our results show that the PTM outperforms other standard machine learning techniques by notable margins. Table (4.2) summarizes the accuracy comparison results.

We evaluated our model on a range of network traffic datasets. These include the well-known LBNL dataset [Allman *et al.*, 2005], collections from West Point and NSA [United States Military Academy, 2009], and University traffic from CU and GMU. For each host, multiple 500-entry segments of Netflow were randomly sampled for testing data, while the rest was used for training. When a dataset can in the form of packet-traces, we converted them into Netflow format.

4.6.1 Datasets

Columbia University dataset The Columbia University dataset consists of traffic captured over the course of three weeks in the March of 2010. The traffic was captured from a SPAN port and covers a significant portion of traffic from the Computer Science depart-

ment, containing over 7 billion packets. Only packet headers over TCP were captured in this dataset. **George Mason University dataset** The George Mason University dataset was similarly captured and contains the traffic from the Computer Science department at that university. Only TCP headers were captured. **University of Michigan/Merritt dataset** was obtained through predict.org [PREDICT, 2011] and contained a single day's worth of traffic from the University of Michigan's networks. The **Lawrence Berkley National Laboratories (LBNL) dataset** comes from their enterprise tracing project [Allman *et al.*, 2005; Pang *et al.*, 2005] and contains traffic that spanned several days across capture in 2004 and 2005. The **West Point dataset** and **National Security Agency (NSA) dataset** were obtained from the same source at [United States Military Academy, 2009]. These are general research datasets containing instances of intrusion-related activity such as port-scans. Given that several datasets only had a day's worth of traffic, we extract day-level subsampling of data from each dataset In order to keep experiments consistent, results over larger datasets such as GMU and CUCS are averages over randomized samplings of days. For each day's traffic, if the dataset was available only in packet format, we performed a packet-to-netflow translation using `nfcapd` and `softflowd` to obtain the flow-layer statistical representation discussed in the previous section. Network traffic is often extremely skewed in terms of volume, where a small number of hosts is responsible for generating the majority of the traffic.

The majority of hosts on the network did not contain more than 1000 flow connections. In order for us to accurately evaluate the performance of our models, thresholds were used to filter out hosts with too little traffic in order to avoid unreliable performance estimations due to underfiting. This entailed a 1,000-flow threshold for the smaller datasets such as NSA and West Point, and a 5,000-flow limit for the larger university datasets. For the larger university datasets, we filtered the data by identifying class-C subnets belonging to that university within the dataset. This is done in order to filter out foreign hosts, not belonging to that network. For example, large amounts of web traffic to google would make that host appear very prominent in the traffic dataset; we are only interested in evaluating hosts belonging to these individual organizations, and not foreign connections. Given that the IP addresses were obfuscated, this estimation was done by observing the distribution

Dataset	Num hosts	Baseline	Gaussian	kNN	PTM
West Point Border	47	2.13%	36.0%	36.5%	**73%**
NSA Capture	45	2.22%	31.1%	34.1%	**45%**
LBNL Enterprise	35	2.86%	49.4%	51.2%	**86%**
Columbia Univ.	82	1.22%	40.5%	40.9%	**86%**
George Mason Univ.	65	1.54%	35.5%	39.6%	**85%**
UMich. Merit	135	0.74%	44.2%	53.0%	**63%**

Table 4.2: Likehood-evaluation-based classification accuracy of comparable models across datasets.

of IP prefixes. Large groups with similar prefix and diverse outbound distributions most likely to represent class C subnets. This was done under the assumption that a prefix-preserving IP transformation was used to obfuscate the addressed. A related study on identifying networks based on IP prefixes in anonymized traffic is provided by [Burkhart *et al.*, 2008]. After filtering in this manner, we can be confident that the hosts we've extracted are representative of their networks as well as contain enough traffic such that the models can be reliably evaluated.

4.6.2 Experiment parameters

Our classification experiment follows standard practice: the model that yielded the highest likelihood on the test sample labels the sample as belonging to that particular class. To understand this notion of accuracy: if there are 100 hosts, a baseline would yield a 1% chance of a correct random guess. Table (4.2) shows the performance of the PT-Markov model. This model consistently achieves significantly higher performance than the baseline, as well as top-performing alternative models in this field. The use of Netflow-like statistics permits training with significantly less data than alternative time-series models such as HMMs, which could require data on the order of days due to sparsity problems (discussed later.) 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, on Columbia's network. This quantifies the degree to which our models are

Dataset	1 - 10	11 - 20	21 - 50	50-100	100-200	200-1000	1000+
LBNL Enterprise	20	5	0	1	0	9	0
NSA Capture	23	1	2	0	1	2	16
West Point Border	27	11	0	0	0	1	8
George Mason Univ.	28	26	10	1	0	0	0
Columbia Univ.	50	28	4	0	0	0	0
UMich. Merit	59	34	27	3	2	10	0

Table 4.3: Distribution of model sizes (number of port-bins) per dataset.

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. These datasets were released for very different purposes and coalescing them into a standard form proved difficult at times. 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 for that particular dataset.

Table (4.3) summarizes the distribution of model sizes observed in these datasets. As we can see, the vast majority of hosts observe very little variance in the amount of services executed. Without model, it's obvious to see which are hosts are port-scanners given their large model size which is based on outbound behavior.

In addition, compared out model to hidden Markov models. To achieve this, we first had to reshape the data into a format that can be used with an HMM. This required delineating the time-series data into segments of network-traffic so that an emissions model for the HMM may be trained. We use the Columbia dataset for this experiment, because unlike the other datasets, it was captured over the course of three weeks and the data was already divided into multiple days, thus a minimum amount of data mangling needed to be done to ensure little influence on the results. We used a data format composed of 288 distinct port histograms, each estimated in the same format as the PTM, 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).

Results comparison with HMM models are given in Table (4.4). "Raw" indicates un-

	Baseline	G-HMM	M-HMM	D-HMM	PTM
Columbia Univ. Raw	1.22%	—	22%	25%	**76%**
Columbia Univ. PCA	1.22%	10%	—	—	—
Columbia Univ. SESSION	1.22%	28%	22%	25%	—

Table 4.4: Classification performance comparison of comparable time-series models on CUCS traffic. Dashed entries indicate infeasible settings. PTM dramatically outperforms HMMs.

processed 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. As Table (4.4) shows, the PTM performs considerably better than the HMM. A range of different types of HMMs were tested, using different emissions models. G-HMM represents HMMs with Gaussian emissions, similar to the PTM, M-HMM represents those with Multinomial emissions, and D-HMM represents those with Discrete Probability Tables emissions. The optimization functions of these HMMs were derived in the same manner as the PTM, except with an additional EM step to estimate the hidden state parameters. As we can see, the PTM outperformed the HMM universally. This result is mainly due to the sparsity problem that plague network traffic. Because we do not always have continuous streams of network data, across multiple days, gaps in the traffic might be implicitly captured as a hidden state by learning algorithm of the HMM. The PTM on the other hand simply takes advantage of the fact that the states are tied with the services, and thus using the port values, we know exactly which state we are in. Thus, the state transition table can be optimally estimated and we can easily use up to several hundred states without any problems.

Concluding remarks

This results contained within this chapter demonstrated the potential for using time-series models for network behavior modeling. While we have had success with the PTM in this research problem, we do expect that in future, research more accurate models will be derived that outperform this model. The value of the kernel-based approach is not bounded by

the specific model proposed, but include the principles behind this construction. Given the large scale of network traffic datasets, designing learning-algorithms for this field encounters rather unique challenges in terms of complexity and scale. Machine learning models must be mindful of the characteristics of network traffic that make it unique to other learning problems. As hidden Markov models results show, intricacies in traffic feature representation and assumptions about latent state behavior can be difficult to establish. The scale of modern network traffic datasets demands attention to this detail when designing new algorithms; training algorithms that require multiple passives over the dataset, or those that scale super-linearly with the data, will face challenges in runtime constraints and possibly suffer under-fitting. The PTM encapsulates behavior in a comparably simple structure, with uncomplicated variable-interactions in an intuitive representation. In the single-Gaussian emissions case, the model is trained on one-pass of over the data. We showed through empirical evaluation how this model can be sufficiently accurate enough to capture the differences in the distinct types of behavior encountered in network traffic.

Chapter 5

Kernel Extensions for Network Behavior Model

Introduction

Having an accurate model for host behavior extends to certain practical applications, such as anomaly detection. However, the main goal of this thesis is to derive methods by which we can measure behavior similarity, in order to perform statistics-preserving mixed-net-based anonymization on the underlaying dataset. This requires the derivation of a measure of similarity between our behavior models. Measuring such similarity has been studied in different contexts in networking. Some representative works include measuring the similarity within packet content distributions for anomaly detection [Cretu *et al.*, 2008], identifying host roles based on communications patterns represented by the traffic dispersion graph [Tan *et al.*, 2003], and a similar study by [Coull *et al.*, 2011] that proposed clustering hosts based on similarity measured by Dynamic Time Warping over flow-level statistics; this latter work is the most closely related study to this thesis and comparisons will be made inline with the explanation of our methods.

The approach described in this thesis is distinguishable from related works in that, rather than comparing data samples directly, the information contained therein is first abstracted by fitting time-series models (PTM) this data, and metrics are derived to compute similarity between the resulting model parameters instead. This metric is implemented as a probability product kernel derived for the PTM. Consider samples $\mathbf{x}, \mathbf{y}$ and let $\phi(\mathbf{x})$ be some non-linear mapping on the components of $\mathbf{x}$, for example $\phi(\mathbf{x}) = [x_1^2, x_2^2, x_1 x_2, ...]$. A kernel function computes the inner product of this mapping in a high dimensional inner-product space ($\mathcal{K}(\mathbf{x}, \mathbf{y}) := \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle$) without having to explicitly calculate $\phi(\cdot)$. In general, kernels are used to capture an assumption of similarity between two data instances. Kernel-based methods typically utilize pair-wise comparisons between data samples, and often take the form of weighted functions such as $h(\mathbf{y}) = \sum_i \alpha_i \mathcal{K}(\mathbf{y}, \mathbf{x}_i) + b$. These algorithms provide several benefits. First, with proper regularization, noise in the data samples may be smoothed during model-parameter estimation rather than being factored into similarity measurement. Second, in time-series distributions, comparing similarity in model parameters is often orders of magnitude faster than comparing the samples themselves since the model parameters are often orders of magnitude smaller. Third, with regard to classification, linearly inseparable samples may be linearly separable in the Hilbert space defined by

the kernel used. Finally, exchanging models for comparison between different organizations is possible, without risk of privacy loss. In addition, by extending our work into the domain of kernel machines, a range of different algorithms becomes available.

This chapter begins with an introduction to various clustering paradigms. One of the main contributions of this thesis is presented, that is the derivation of the probability product kernel for the PTM, along with derivations of new clustering/matching algorithms in a kernel-based framework. The choice of this approach was motivated by our previous work in time-series clustering [Jebara *et al.*, 2007].

5.1 Paradigms in time-series data clustering

Clustering time-series data presents unique challenges over the more familiar case where data samples are points embedded in a vector space. Time-series data often represent the output of some stochastic process, and these samples are often not aligned and vary in length. This property is a fitting characterization of network traffic, where instances from different hosts may be captured at different times, with different durations.

There are three main paradigms in time-series data clustering, representing different extremes, and comparisons between related approaches are discussed within this section. These approaches span the continuum in the theory of time-series clustering. At one end of the continuum are data-driven methods for similarity comparison. These methods typically compare evaluate similarity in the time-series sequence based on pair-wise subcomponent similarity. As an example, given two sequences $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M\}$ and $\mathbf{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_N\}$, one can simply form a $M \times N$-sized matrix of pair-wise similarity values measured by the inner product where $S[i, j] = \langle \mathbf{x}_i, \mathbf{y}_j \rangle$. The aggregate similarity between the two sequences can be solved as the average of the matrix $S(\mathbf{X}, \mathbf{Y}) = \frac{1}{MN} \sum_{i,j} S[i, j]$. this metric, of course, runs in polynomial time $O(MN)$. A representative example of this technique is Dynamic Time Warping (DTW). Variations on this technique have been proposed, such as those that measure only components of the similarity matrix near diagonal; this translates to comparing data within a short window of each other in time. This variation of DTW is used for flow-record similarity measurement in the study by [Coull *et al.*,

2011]. Fundamentally, the DTW-class of algorithms is data-bound. Given a large network trace, with thousands of hosts and millions of flow-records, the computational cost of such algorithms can easily make this approach impractical.

5.1.1 Soft-clustering via expectation-maximization

The next step along the continuum is to construct a hierarchical probability model, often in the form of a mixture-model. Optimization of this model would then yield an optimization of the parameters of the individual clusters as an intrinsic property. A simply construction typically consists of assuming a linear mixture where the weights are learned through an iterative gradient-descent procedure. Let $p(\mathbf{X}|\theta)$ represent the probability of observing $\mathbf{X}$ for a distribution p with model parameter θ. A mixture extension with M clusters would then be:

$$p^*(\mathbf{X}|\Theta) = \sum_{i=1}^{M} \pi_i p(\mathbf{X}|\theta_i) \qquad \pi = \{\pi_1, \pi_2, ..., \pi_M\}, \quad \sum_i \pi_i = 1.$$

After optimizing $p^*(\mathbf{X}|\Theta)$, by taking the gradient with respect to the parameters Θ and setting to zero, and training the model over the data, the individual sub-models $\theta_1, ..., \theta_M$ then represents the parameters of the individual clusters. Expectation maximization is referred to as a "soft" clustering because the influence of each data sample within the parameter update rule is weighted by its posterior probability to that sub model. The K-Means algorithm is the "hard" counterpart to EM, and the optimization is identical with the exception that no weight is used and the sample is simply assigned to the cluster with the higher posterior probability at each iteration.

The EM-based approach to clustering is more model-driven than the previous DTW approach, in that a mixture model is used to represent the data. However, multiple passes over the data is still necessary when using EM. Specifically, the dataset needs to be evaluated $O(cL)$ where L is the total length of all of the data samples, and c (where $c << L$) is the number of iterations needed for the algorithm to converge. The downside of using an EM approach is that the algorithm makes parametric assumptions about the dataset; fundamentally, that it is distributed in a mixture model – this assumption does not always hold, especially in the case of network data where the distributions can be very non-parametric.

5.1.2 Graph-based clustering methods

Finally, the graph-based approach to clustering represents the other extreme. In this setting, separate models are trained on each data instance, and these models are then treated as nodes within a graph. The edge weight $e_{i,j} \in E$ between two models θ_i, θ_j is calculated using some measure of similarity between the two nodes. Typically a kernel is used for this similarity measurement. Clustering corresponds to recovering the partition for the graph that yields the minimum-weight edge cut. Additionally, given the fact that the distribution of edge weights can imbalance the cut, the Normalized Ratio cut, or normalized-cut for short, is more often used. While provably optimal, recovering the normalized-cut solution for graph partitioning is a problem that has been proven to be NP-Complete [Shi and Malik, 2000]. In practice, spectral graph partition is often used to approximate the normalized-cut by using the eigenvectors of the graph's Laplacian matrix, which is the normalized affinity matrix of pair-wise similarities. The solution to the partition problem is then recovered by the vector $\mathbf{q}$:

$$\arg\max_{\mathbf{q}} \mathbf{q}^\top \mathbf{L}\mathbf{q}, \quad \mathbf{q} = [q_1, q_2, ..., q_N], q_i \in \{-1, +1\} \tag{5.1}$$

Solving for $\mathbf{q}$ is easily recognizable as the eigenvector-decomposition problem. The graph-cut method has the benefit of being the most efficient of all three approaches, both in terms of runtime and memory costs. Further, this approach remains agnostic to the underlying distribution of the data. For sparse data where the dimensionality of the feature set is large, avoiding such parametric assumptions can often lead to improvements in accuracy over methods such as EM, as shown in the following works [Kannan *et al.*, 2000; Ng *et al.*, 2001; Jebara *et al.*, 2007; Tony Jebara, 2007].

5.2 Derivation of the probability product kernel for the PTM

In this thesis, we take an approach that falls within these extremes. Rather than comparing the data directly, as in DTW, or by optimizing a model over the entire sample distribution, we fit a PTM model to represent each host – trained on that host's traffic – then use spectral clustering to find the optimal partition among these hosts. Therefore, a kernel

between PTM models is necessary. This section discusses the derivation of such a kernel.

Typically, kernels are used for vector data $\mathbf{x} \in \mathbb{R}^d$. However, a special class of kernels exist which operate over distributions on $\mathbf{x}$. $\mathcal{K}(p, p') = \langle p(\mathbf{x}|\theta), p(\mathbf{x}|\theta') \rangle$. For brevity, we denote $p(\mathbf{x}|\theta)$ as p and $p'(\mathbf{x}|\theta')$ as p'. An instance of this class is the probability product kernel (PPK) [Jebara *et al.*, 2004]. The PPK is a generalized kernel that computes the inner-product of two distributions over the space of all possible data inputs $\mathbf{x}$:

$$\mathcal{K}(p, p') = \int_{\mathbf{x}} p^{\beta}(\mathbf{x}|\theta) p^{\beta}(\mathbf{x}|\theta') d\mathbf{x}. \tag{5.2}$$

When applied to vectors, this kernel has an unusual interpretation – it calculates the similarity between two data points defined by a function over the pair of distributions fit to these single points. However, in the time-series domain, where models such as HMMs are frequently used to concisely represent single sequences of time-series data, this is not only a meaningful interpretation but a computationally efficient approach to measuring similarity. The PPK is closely related to other similarity metrics in machine learning. When $\beta = 1/2$ in equation (5.2), the kernel affinity is equivalent to the well-known Bhattacharyya affinity between two probability distributions. This affinity is related to the well-known Hellinger divergence $H(p, p') = \frac{1}{2} \int_{\mathbf{x}} \left(\sqrt{p(\mathbf{x})} - \sqrt{p'(\mathbf{x})} \right)^2 d\mathbf{x}$ according to the following relation $H(p, p') = \sqrt{2 - 2\mathcal{K}(p, p')}$.

The PPK is solvable in closed-form for a range of distributions. These include the multivariate-Gaussian distribution, Gaussian mixture model, multinomial distribution, hidden Markov models with various emissions distributions, as well as continuous state-space time-series models such as the linear dynamical system, the details of which can be found in [Jebara *et al.*, 2004]. Having a elegant closed-form solution to a metric with a rich interpretation makes this an attractive method with which to measure time-series similarity.

This section shows the derivations for the probability product kernel for the protocol-transition Markov model. The details of PPK-construction is described in [Jebara *et al.*, 2004]. At its core, the PPK has two components: the affinity in the emissions model, and the coupling of the state estimates. In the first component, we derive what is referred to as the elementary kernel $\Psi(\cdot)$, which is the affinity between the emissions models for p and p' integrated over the space of all emissions $\mathbf{x}$.

$$\Psi(\theta, \theta') = \int_{\mathbf{x}} p^{1/2}(\mathbf{x}|\theta) p^{1/2}(\mathbf{x}|\theta') d\mathbf{x}. \tag{5.3}$$

In order to derive the PPK for the PTM model, the elementary kernel of (5.3) needs to be solved using the PTM mixture-of-Gaussians model. This can be solved by expanding the product and following the same complete-the-squares technique as described in [Jebara *et al.*, 2004]. Due to the linearity of integration, the elementary-kernel of a linear-mixture distribution can be solved as a function of pair-wise sub-elementary-kernels $\widehat{\Psi}$, on the sub-models of the mixture. This kernel is derived here.

The main difference between the kernel for the PTM model and the HMM model is that the states for the PTM are fully observed. For the HMM, the elementary kernel between the pair of emissions distributions involves integrating over all of the sub-models. For the PTM, the state is tied to the port value, therefore the set of common states between two models is observable, and the integration can be restricted to only these common sub-models. Let $\mathcal{C}$ represent the set of common states between two hosts. Let $(\mu_{i,m},\ \sigma_{i,m})$ represent the sufficient statistics of the m^{th} sub-model of the i^{th} Gaussian mixture, then we have the following:

$$\begin{aligned}
\mu^{\dagger} &= \mu_m/\sigma_m + \mu_n/\sigma_n && (5.4)\\
\sigma^{\dagger} &= (\sigma_m^{-1} + {\sigma_n}^{-1})^{-1} && (5.5)\\
\mathbf{Z} &= \mu_m \sigma_m^{-1} \mu_m + \mu_n {\sigma_n}^{-1} \mu_n - \mu^{\dagger}\sigma^{\dagger}\mu^{\dagger} && (5.6)\\
\widehat{\Psi}(\theta_m, \theta_n) &= \frac{\sqrt{\sigma^{\dagger}}}{(\sigma_m\sigma_n)^{1/4} \exp\left(-\frac{1}{4}\mathbf{Z}\right)} && (5.7)\\
\Psi(i,j) &= \sum_{m\in\mathcal{C}} \sum_{n\in\mathcal{C}} \alpha_m \gamma_n \widehat{\Psi}(\theta_{i,m}, \theta_{j,n}). && (5.8)
\end{aligned}$$

α, γ represent the mixture coefficients within the i, j models, 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.* $\sum_i \alpha_i = 1$). In practice, the size of each PTM model is small (c.f. Table (4.3)) and the amount of overlap between any two given hosts ($\mathcal{C}$) is smaller still – this translates to a very fast and efficient kernel for comparing model similarity.

The essence of the probability product kernel is that all possible configurations of settings

PTM 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 $t = 1 \dots T$

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

end

$\mathcal{K}(p, p') = \sum_{q_T} \sum_{q_T'} \Phi(q_T, q_T') \Psi(q_T, q_T')$

Table 5.1: Fast iterative algorithm for the probability product kernel for the PTM.

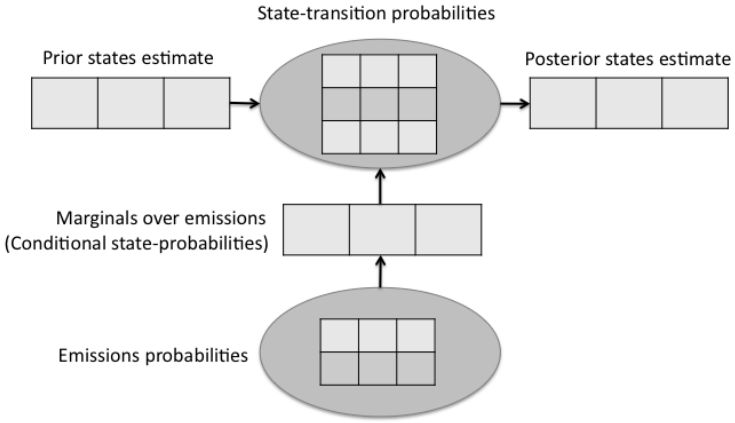


Figure 5.1: One step in the Junction Tree algorithm.

are evaluated in an efficient manner. Even though the states of the PTM model are observed, when computing the kernel between two models, we still evaluate over all possible pair-wise configurations. This means that the coupling of the states is similar to the one found in the PPK for the HMM. Once the elementary kernel $\Psi(\theta, \theta')$ is obtained, marginalizing over the hidden states is done using the Junction-Tree factorization which yields the efficient dynamic-programming solution shown in Table (5.1). This function recovers the kernel between two PTM models in $\mathcal{O}(TMN)$ operations where T is the length of the sequence and M and N are the sizes of the models (cf. Table (4.3)).

However, the PTM and HMM have different underlying interpretations, which can be leveraged to obtained an even more efficient kernel based on the PPK. Consider each iter-

ation of the Junction Tree algorithm as shown in Figure (5.1). Probability estimation for HMMs requires marginalization over the hidden states, and factoring in both the emissions probabilities and the prior state estimate. This can be interpreted as estimating a posterior estimate for the hidden state at each time step given the evidence observed from prior emissions and state estimates. Written in vector notation this takes on the following form:

$$\mathbf{q}_t = \text{diag}(\mathbf{p}_t)\mathbf{A}^\top \mathbf{q}_{t-1} \tag{5.9}$$

where $\mathbf{p}_t$ represents the marginalized emissions probabilities at time-step t. Here is where the difference between the two models can be leverage to our advantage. Given that the state values for the PTM is observed and does not depend on the emissions probability, the posterior state estimate can be calculated by dropping the $\text{diag}(\mathbf{p}_t)$ term. Thus the state values take on the following linear relation:

$$\mathbf{q}_T = \prod_{t=1}^{T} \mathbf{A}^\top \mathbf{q}_{t-1}. \tag{5.10}$$

This is a familiar function in linear dynamical systems, the continuous-state analog of HMMs, and familiar theories are applicable. In particular, the Lyapunov steady-state condition for linear state-space models shows that equation (5.10) is exponential stable, *i.e.* converges, if the real components of the eigenvalues of $\mathbf{A}$ are less than 1. Given that $\mathbf{A}$ is a row-normalized transition matrix ($\sum_j A[i,j] = 1$ for all i), the eigenvalues for $\mathbf{A}$ can never be greater than 1, thus the above equation converges. In practice, convergence typically happens after 10 iterations – this can be the value set for T in the probability product kernel. Since the emissions model does not factor into the state estimate, we can solve for the convergent state estimate first, then resolve the emissions. This produces the following for efficient pseudo-PPK:

In the algorithm given in Table (5.2), the state estimates are calculated independently of the emissions model. Note that the matrix exponential is used. In practice, it was found that convergence typically occurs after ten iterations, therefore $T = 10$ is an accurate setting. This kernel essentially calculates the PPK, not over the entire state space, but rather the most likely setting for the state estimate – which makes this a maximum-a-posterior (MAP) estimate for the PTM probability product kernel. The benefit of this kernel is that it is

MAP-estimate for the PTM probability product kernel $\mathcal{K}(p, p')$:
$\mathbf{q}_0, \mathbf{q}'_0$ are the initial state estimates for the two models p, p'.
$\widehat{\mathbf{q}_T} = (\mathbf{A}^\top)^{T-1}\mathbf{q}_0$
$\widehat{\mathbf{q}'_T} = (\mathbf{A}^\top)^{T-1}\mathbf{q}'_0$
$\mathcal{K}(p, p') = \sum_{\widehat{\mathbf{q}_T}} \sum_{\widehat{\mathbf{q}'_T}} \Phi(\widehat{\mathbf{q}_T}, \widehat{\mathbf{q}'_T}) \Psi(\widehat{\mathbf{q}_T}, \widehat{\mathbf{q}'_T})$

Table 5.2: Faster iterative algorithm for the probability product kernel for the PTM using the maximum-a-posteri estimate. $T = 10$ is a good setting for convergence.

fast, memory efficient, and simple to implement, and achieves identical performance with the full PPK, while being less likely to suffer from numerical problems given that the emissions models are not factored into the calculation at each iteration. Just like the earlier model, $\Phi(q_T, q'_T)$ is an $M \times N$ matrix where M, N are the sizes of the i, j models, respectively – which is the number of unique ports observed, and the actual summation occurs over the common states, which is often a smaller subset.

5.2.1 Isotropic emissions model

$$\mathcal{K}(p, p') = \frac{1}{\sqrt{(4\pi\sigma^2)}} \exp\left(-||\mu - \mu'||^2/(4\sigma^2)\right). \tag{5.11}$$

In the case of a single Gaussian emissions model for the PTM, a simpler form of the probability product kernel is available. Under the setting $\beta = 1$, and fixing the variance in the two separate distributions for p, p', the elementary kernel for Gaussian model is conditioned only on the μ parameter of the distribution. This form is shown in the above equation. This reduced elementary kernel is even faster to evaluate, and could prove more stable in the cases where certain states have too few samples for the learning algorithm to estimate the variance parameter σ accurately.

5.2.2 Mutual information kernel

Another kernel for the PTM is available using the mutual-information kernel described in [Yin and Yang, 2005]. Shown in equation (5.12), this kernel calculates the Hilbert-space inner-product as a function of the cross-likelihood values from the two densities. Let $\theta_{\mathbf{x}}, \theta_{\mathbf{y}}$

represent the models trained over samples $\mathbf{x}, \mathbf{y}$, respectively. This kernel takes the following form:

$$\mathcal{K}(\theta_{\mathbf{x}}, \theta_{\mathbf{y}}) = \exp\left\{ -\frac{||p(\mathbf{y}|\theta_{\mathbf{x}}) - p(\mathbf{x}|\theta_{\mathbf{y}})||}{2\sigma^2 p(\mathbf{x}|\theta_{\mathbf{x}}) p(\mathbf{y}|\theta_{\mathbf{y}})} \right\}. \tag{5.12}$$

The main benefit of this mutual-information kernel is the ease of implementation, requiring only a set likelihood evaluations per model pair. The downside of this derivation is the overhead incurred in having to explicitly calculate 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.

5.3 Spectral algorithms for PTM-measure host behavior

Given the PPK, a range of kernel-based methods are available. Among the most useful is spectral clustering, which is a graph-based clustering algorithm is robust when used on datasets whose distributions do not follow classic parametric distributions such as Gaussians. This section describes how to perform spectral clustering of host behavior using the PTM and the PPK. In addition, we propose a new spectral partitioning algorithm which produces more stable and balanced clusters.

5.3.1 Unbalanced spectral clustering with the PPK

The spectral approach to time-series clustering involves estimating a PTM model for each sequence. The PPK is then computed between all pairs of PTMs to generate a Gram matrix which is used for spectral clustering. This approach leverages both parametric and non-parametric techniques in the clustering process; parametric PTMs make some assumptions about the structure of the individual sequences (such as Markov assumptions) but the spectral clustering approach makes no assumptions about the overall distribution of the sequences (for instance, *i.i.d* assumptions). Empirically, this approach (Spectral Clustering of Probability Product Kernels or SC-PPK) achieves a noticeable improvement in clustering accuracy over fully parametric models such as mixtures of PTMs or naive pairwise likelihood comparisons.

Listed below are the steps of our proposed algorithm. Spectral clustering is an eigenvalue-relaxation of the NP-hard normalized-cuts graph segmentation problem described by Shi & Malik [Shi and Malik, 2000]. Our implementation of SC-PPK is a time-series analogue of the Ng & Weiss spectral clustering algorithm presented in [Ng *et al.*, 2001].

The SC-PPK algorithm:

1. Fit a PTM to each of the $n = 1 \ldots N$ 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 matrix: $L = D^{-1/2} A D^{-1/2}$.
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 PTM models.

Figure (5.2) shows the results of the spectral clustering algorithm over the Columbia dataset. The DNS names are printed in order to provide some measure of accuracy. While by no means are they definitive labels for true behavior, similar DNS names do provide some measure of confidence in the accuracy of the algorithms when similarly labeled hosts are grouped into the same cluster. These hosts are mostly server machines, many were configured and operated by the IT department within the Computer Science department, therefore they are consistently configured. The "web1,web2,web3,web4" machines, for example, are web-servers behind a load-balancer; it is not surprising to see them grouped together. The "sos" machines are general purpose computing machines belonging to the Network Security lab; they are used by different users and their behavior is not universally similar.

(1) ng911-db2	(3) manata	(6) web3	(8) unknown
(1) forest	(3) metrorail	(6) web4	(8) zapp.win
	(3) raphson	(6) web3157	
(2) dhcp11	(3) metro20	(6) game	(9) metro23
(2) corsica		(6) honda	(9) goya
(2) dhcp25	(4) kathy-pc.win	(6) cs	(9) uribe
(2) dhcp30	(4) kkh	(6) animal	(9) tardieu-desktop
(2) dhcp49	(4) sekhmet	(6) chihiro	(9) boyacipc2
(2) workstudylaptop	(4) lagrange	(6) raphael	(9) irtdesk1
(2) racecar	(4) klimt	(6) gabriel	(9) picard
(2) leaf	(4) pyromania	(6) templar	(9) cityhall
(2) spectral	(4) bailey		(9) multitouch
(2) fdr		(7) walker.win	(9) ense1
(2) water	(5) dhcp64	(7) carter.win	
(2) coulomb03	(5) dhcp72	(7) baker.win	(10) irtdesk5
(2) dhcp85	(5) dhcptemp27	(7) fuji	(10) guajira
(2) furst-pc	(5) dhcptemp33		(10) hamming
(2) apadana	(5) dhcptemp48	(8) ense-client	(10) green
(2) unknown	(5) mail	(8) sos2	(10) sos1
	(5) fry.win	(8) sos5	(10) sos6
(3) neale.ccls		(8) ense2	(10) sos9
(3) pachira	(6) web1	(8) cutunes	(10) elba
(3) cluster00.ncl	(6) web2	(8) hikari	

Figure 5.2: Results of spectral clustering with 10 clusters.

5.3.2 Balanced spectral partition with the PPK

Notice that spectral clustering algorithm is not guaranteed to produce balanced clusters, or clusters with a minimum size. This is because the clustering is based on an eigenvalue approximation of the normalized minimum-weight graph cutting problem [Shi and Malik, 2000]. As such, the distribution of the cut sizes are entirely determined by the dataset, and the pair-wise affinities (edge weights) between the data samples. An unevenly distributed dataset would yield to unbalanced clusterings, and distributions with very anomalous behavior could lead to undersized or singleton clusters.

The goal of this thesis, however, is to use clustering in order to provide anonymity to hosts on a network. Therefore, undersized clusters, and clustering without a guarantee of minimum size would not be a complete solution. Nevertheless, spectral clusterings reflects the distribution information within a dataset, and the distribution of cluster sizes do provide useful information about the behavior profiles of the network. In a later section § 7.2.2 we demonstrate how to patch spectral clustering with an hierarchical merging step in order to provide a minimum cluster size guarantee while preserving the distribution information. In this subsection, we present a simpler, more intuitive algorithm that not only guarantees a minimum clustering size, but also a balanced clustering.

Spectral-Partition provides the pseudocode for this algorithm. Spectral partition is based on half-space partitions, where the data space is continuously divided in half, recursively – until the minimum cluster size is met. This is a stable algorithm with input and output invariance. The algorithm recursively partitions the space of the dataset using the trained model parameters represented by Θ. At each layer of the recursion, the Fiedler vector is extracted from the eigenvectors of the Gram matrix's graph Laplacian, which is the eigenvector corresponding to the second largest eigenvalue – the vector corresponding to the largest eigenvalue is the vector of all ones. The pivot point is calculated at the half-way point after sorting the components of the Fielder vector. Conceptually, this partition is the approximate optimal graph cut that balances the normalized sum of the edge weights between the two clusters. The partitioning is tracked by applying the same partitions to the original data index, which is simply a vector of indices of length N $[1, 2, 3, \ldots, N]$. The Mergelabels concatenates the retrieved label and index vectors and ensures proper order-

ing is maintained. When merging two partitions, the label vectors are $1 \times k$-sized vectors that hold the cluster label for each partition $[1, 1, ..., 1], [2, 2, .., 2], \ldots, [N/k, N/k, .., N/k]$. The function resolves label collisions by incrementing the label of one of the clusters.

SPECTRAL-PARTITION($\Theta, index, k$)

1 $d \leftarrow$ LENGTH(Θ)
2 $\triangleright$ Check if we have reached the minimum cluster size
3 **if** $d \leq k + 1$:
4 **then return** $\{[1, 1, \ldots, 1]_{1\times d}, index\}$
5 $A \leftarrow$ CALCULATEGRAMMATRIX(Θ, Θ)
6 $D_{m,m} \leftarrow \sum_n A_{m,n} \forall m = 1, \ldots, N$
7 $L \leftarrow D^{-1/2} A D^{-1/2}$
8 $[V, D] \leftarrow$ EIGENVECTORDECOMPOSITION(L)
9 $\triangleright$ Sort the eigenvectors contained in V based on the eigenvalues
10 $[D, \text{idx}] \leftarrow$ SORT-DESCENDING(D)
11 $\triangleright$ Take the vector corresponding to the second largest eigenvalue – this is the Fiedler vector
12 $V \leftarrow V[:, \text{idx}(2)]$
13 $\triangleright$ Sort the components of the Fiedler vector and find the splitting point
14 $[V, \text{idx}] \leftarrow$ SORT-ASCENDING(V)
15 $p \leftarrow \lfloor d/2 \rfloor$
16 $idx_1 \leftarrow \text{idx}[1, \ldots, p]$
17 $idx_2 \leftarrow \text{idx}[p + 1 : d]$
18 $index_1 \leftarrow \text{index}[idx1]$
19 $index_2 \leftarrow \text{index}[idx2]$
20 $[label1, index_1] =$ SPECTRAL-PARTITION($\Theta[idx1], index_1, k$)
21 $[label2, index_2] =$ SPECTRAL-PARTITION($\Theta[idx2], index_2, k$)
22 **return** MERGELABELS(label1,ind1,label2,ind2)

Figure (5.3) shows the results of the spectral partition function on the same dataset, with a minimum cluster size set to 5. Here, we see the balanced partitioning and the minimum cluster sizes preserved. Many similarly-named hosts appear in the same clusters, which

(1) web1	(5) dhcptemp48	(9) metro20	(13) green
(1) web2	(5) neale.ccls	(9) metro23	(13) sos1
(1) web3	(5) pachira	(9) goya	(13) sos2
(1) web4	(5) cluster00.ncl	(9) uribe	(13) sos5
(1) web3157	(5) manata	(9) tardieu-desktop	(13) sos6
(2) game	(6) metrorail	(10) boyacipc2	(14) sos9
(2) honda	(6) raphson	(10) irtdesk1	(14) ense1
(2) cs.columbia.edu	(6) dhcp11	(10) irtdesk5	(14) ense2
(2) animal	(6) corsica	(10) coulomb03	(14) cutunes
(2) chihiro	(6) dhcp25	(10) dhcp85	(14) hikari
(3) raphael	(7) dhcp30	(11) furst-pc	(15) lagrange
(3) gabriel	(7) dhcp49	(11) ng911-db2	(15) klimt
(3) templar	(7) workstudylaptop	(11) picard	(15) pyromania
(3) walker.win	(7) racecar	(11) cityhall	(15) bailey
(3) carter.win	(7) leaf	(11) forest	(15) apadana
(4) baker.win	(8) kathy-pc.win	(12) sekhmet	(16) elba
(4) dhcp64	(8) spectral	(12) guajira	(16) unknown
(4) dhcp72	(8) fdr	(12) hamming	(16) unknown
(4) dhcptemp27	(8) water	(12) multitouch	(16) mail
(4) dhcptemp33	(8) kkh	(12) ense-client	(16) zapp.win
	(8) fuji		(16) fry.win

Figure 5.3: Results of spectral partitioning. The minimum cluster size was set to 5.

adds confidence to the accuracy of this algorithm. While using the spectral partitioning function does provide minimum-cluster-size guarantees, it does so at some cost to accuracy. Consider the case of where three balanced clusters are equally distributed with distances equal to each other. The first cut will split the dataset along the middle of one of these clusters, which is a sub-optimal solution. In general however, such cases are rare in network behavior. The choice of which algorithm to use, and how to use it with respect to minimum cluster sizes, will depend on the distribution of the dataset and the anonymization policy of the source provider.

5.4 Accuracy and runtime evaluations

The anonymization algorithms presented in this thesis is conditioned on two things: the accuracy of the model in capturing the representation of host behavior and the accuracy in the kernel metric is measuring the similarity between two models. As such, the much of the experiments focus on empirical evaluations of the accuracy of these two components. In previous sections, accuracy comparisons of the PTM versus other similar models have been presented. This section shows the results of comparing the performance of the PTM with the probability product kernel against other methods with similar purpose. Specially, we focus on the clustering techniques based on EM and a similar spectral clustering technique presented by [Yin and Yang, 2005]. Our recent research has shown that these two methods are the closest competitors to our algorithm in general performance on time-series classification problems with other kernels, both in terms of accuracy and runtime [Jebara *et al.*, 2007].

Normally, evaluating the results of clustering is not straightforward. Many factors need to be considered when more than two clusters are evaluated. These factor include measurements on different types of matching-fidelity such as homogeneity, recall, *etc.* For a more detailed description of this problem refer to [Meila, 2007]. A simpler way of measuring clustering-accuracy is to use a two-class clustering set-up where the cluster labels translate directly to classification labels. This presents a simple and reliable estimation of the accuracy of the kernel. This method of evaluation was used in our related paper on clustering

Dataset	EM	SC-MIK	SC-PPK
'cityhall' vs 'racecar'	54%	70%	**100%**
'leaf' vs 'sos1'	64%	80%	**100%**
'cutunes' vs 'racecar'	66%	50%	**94%**
'dhcp49' vs 'sos9'	74%	66%	**94%**
'baily' vs 'cluster00.ncl'	86%	66%	**94%**
'ense-client' vs 'workstudylaptop'	82%	54%	**94%**
'sos6' vs 'cityhall'	74%	54%	**94%**
'picard' vs 'sos9'	**94%**	52%	92%
'leaf' vs 'ense-client'	82%	80%	**92%**
'tardieu-desktop' vs 'ense2'	82%	62%	**92%**
'ense2' vs 'sos2'	82%	50%	**92%**
'corsica' vs 'pyromania'	88%	66%	**86%**
'ng911-db2' vs 'multitouch'	**94%**	74%	80%

Table 5.3: Clustering accuracy comparison between samples from classes of different behavior.

with time-series models [Jebara *et al.*, 2007].

Figure (5.3) shows the comparison of inter-class classification performances.

Exact runtime comparisons are presented in Table (5.4). Evaluations were performed using synthesized traffic samples of 10,000 entries in length with 10 model states; parameters that are set to represent the average available traffic sample for a particular host. As the results show, both the EM-based approach and the spectral clustering approach with the mutual-information kernel exhibited poor performance compared to the spectral clustering algorithm using the PPK. This is because these two former methods are derived using likelihood evaluations over the data and thus their performance scales with the size of the dataset. Conversely, the SC-PPK algorithm runtime grows according to number of hosts and the size of the models. In practice, large enterprise networks may contain tens of thousands of hosts each with hundreds of thousands or millions of flow entries. Clustering methods that are data-bound will have difficulty scaling to that size, whereas the SC-PPK

# OF TIME-SERIES SAMPLES	EM	SC-MIK	SC PPK
10	25s	8.4s	**1.0s**
20	52s	35s	**3.3s**
30	88s	77s	**7.6s**
40	101s	139s	**13s**
50	126s	215s	**21s**
60	153s	311s	**29s**
70	181s	433s	**41s**
80	206s	561s	**52s**
90	230s	704s	**69s**
100	255s	871s	**84s**

Table 5.4: Average runtimes of time-series clustering algorithms. Includes model-training time.

algorithm can handle such scale by the nature of its design. In our approach, each model can always trained independently in linear time, and each element of the similarity matrix can be evaluated independently, the complexity of which is conditioned only on the model size, which is directly tired to the diversity of observed traffic for a particular host. This independency makes parallel processing simple to implement, and efficient solvers for large graph cutting problems are available.

Figure (5.4) shows the scalability of the two faster kernel-based approaches. The SC-PPK method maintains superior runtime and scales favorably compared to the mutual-information kernel presented by [Yin and Yang, 2005]. This feature is due to the fact that the PPK does not require exact computation of the likelihood values for each kernel estimate, unlike the mutual-information kernel, and the EM-based approach. By not using the data samples themselves, we use a dataset that is orders of magnitude smaller when computing the Gram matrix. As such our algorithm runs orders of magnitude faster than our closest competitor asymptotically. Further results from previously published work confirms this property [Jebara *et al.*, 2007].

We can further evaluate the accuracy of the PPK by using it to extract the nearest-

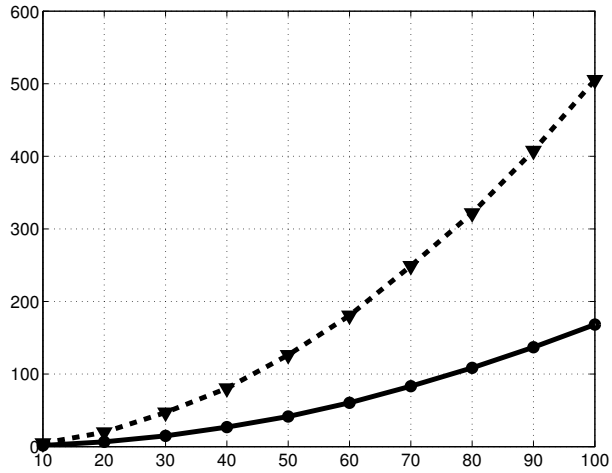


Figure 5.4: Runtime scaling comparison of top two kernel-based clustering methods: mutual-information kernel vs probability-product kernel. Dashed line: Mutual Information kernel, solid line: PPK.

[1] green.cs.columbia.edu	[2] animal.cs.columbia.edu	[3] carter.win.cs.columbia.edu	[4] ense1.cs.columbia.edu
[1] hamming.cs.columbia.edu	[2] baker.win.cs.columbia.edu	[3] corsica.cs.columbia.edu	[4] forest.cs.columbia.edu
[1] leaf.cs.columbia.edu	[2] cityhall.cs.columbia.edu	[3] dhcp64.cs.columbia.edu	[4] green.cs.columbia.edu
[1] metro20.cs.columbia.edu	[2] cluster00.ncl.cs.columbia.edu	[3] dhcptemp33.cs.columbia.edu	[4] leaf.cs.columbia.edu
[1] raphson.cs.columbia.edu	[2] cs.columbia.edu	[3] irtdesk1.cs.columbia.edu	[4] metro23.cs.columbia.edu
[1] web1.cs.columbia.edu	[2] dhcp25.cs.columbia.edu	[3] klimt.cs.columbia.edu	[4] multitouch.cs.columbia.edu
[1] web2.cs.columbia.edu	[2] dhcp72.cs.columbia.edu	[3] sos5.cs.columbia.edu	[4] raphael.cs.columbia.edu
[1] web3.cs.columbia.edu	[2] dhcp85.cs.columbia.edu	[3] sos6.cs.columbia.edu	[4] sos1.cs.columbia.edu
[1] web3157.cs.columbia.edu	[2] neale.ccls.columbia.edu	[3] sos9.cs.columbia.edu	[4] sos2.cs.columbia.edu
[1] web4.cs.columbia.edu	[2] templar.cs.columbia.edu	[3] walker.win.cs.columbia.edu	[4] sos6.cs.columbia.edu

Figure 5.5: Nearest-neighbor clusters are recovered with the CUCS dataset using the PPK.

neighbors in the space of all behaviors present within the dataset. Figure (5.5) shows the output of a nearest neighbor grouping. 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 web servers (web1, web2,...) and compute servers (sos1,sos2,...).

5.5 Behavior visualization and low dimensional embedding

All data – scalar, vector, time-series, or otherwise – can be thought to exist as single points in some high (possibly infinite) dimensional space. "Embedding" algorithms is a class of algorithms designed to capture a representation of these data points in a lower dimension (typically 2 or 3) while maintaining the pair-wise distances in the individual data points. These techniques permit visualization of high dimensional datasets and can help confirm clustering results as well as provide insides into the distribution of the data and the number of appropriate clusters.

5.5.1 Kernel PCA

For most of these algorithms, the only requirement is some measure of data similarity. In the simplest case we consider kernel principle component analysis (KPCA). KPCA for time-series data is a straightforward extension given the derivation of the PPK. These algorithms may be useful in setting where exchanging data between organizations is not desired. Instead, models and low-dimensional vector-space embeddings of data is to be exchanged. The dimensionality reduction in this step is analogous to the previously mentioned PCA. Whereas principal component analysis finds a linear mapping 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 function Φ induced by the kernel function.

$$\mathbf{C} := \frac{1}{m} \sum_{i=1}^{m} \Phi(x_i)\Phi(x_i)^T. \tag{5.13}$$

Since it can be proven that the eigenvector spans the Φ-images of the training data,

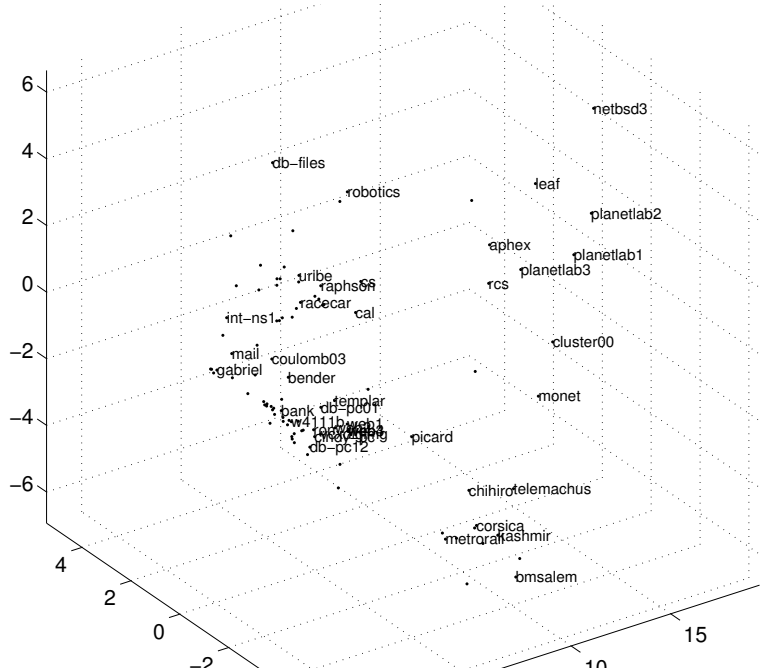


Figure 5.6: Kernel-PCA embeddings for CUCS dataset using HMM models.

the solution set $\mathbf{v}$ takes the form $\mathbf{v} = \sum_{i=1}^{m} \alpha_i \Phi(x_i)$, which leads to the same eigenvalue decomposition problem:

$$m\lambda\boldsymbol{\alpha} = \mathbf{G}\boldsymbol{\alpha}, \qquad \mathbf{G}[i,j] := \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) \tag{5.14}$$

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

$$\langle \mathbf{v}(n), \Phi(\mathbf{x}) \rangle = \sum_{i=1}^{m} \boldsymbol{\alpha}_n(i) \mathcal{K}(\mathbf{x}_i, \mathbf{x}) \tag{5.15}$$

Where the kernel function $\mathcal{K}(\cdot,\cdot)$ can be either the probability product kernel or the cross-likelihood kernel. The matrix of such kernel values $\mathbf{K}$ is known as the Gram matrix.

Figures (5.6) and (5.6) show an examples of lower-dimensional projection using KPCA. In each case, time-series models were fit to the network traffic of these hosts and the proper kernel functions were used to compute the projections according to the equations described above. Results were calculated using distance metrics over HMMs in figure (5.6) and PTMs in figure (5.7). Certain clusters of behavior immediate stands out, such as the planetlab

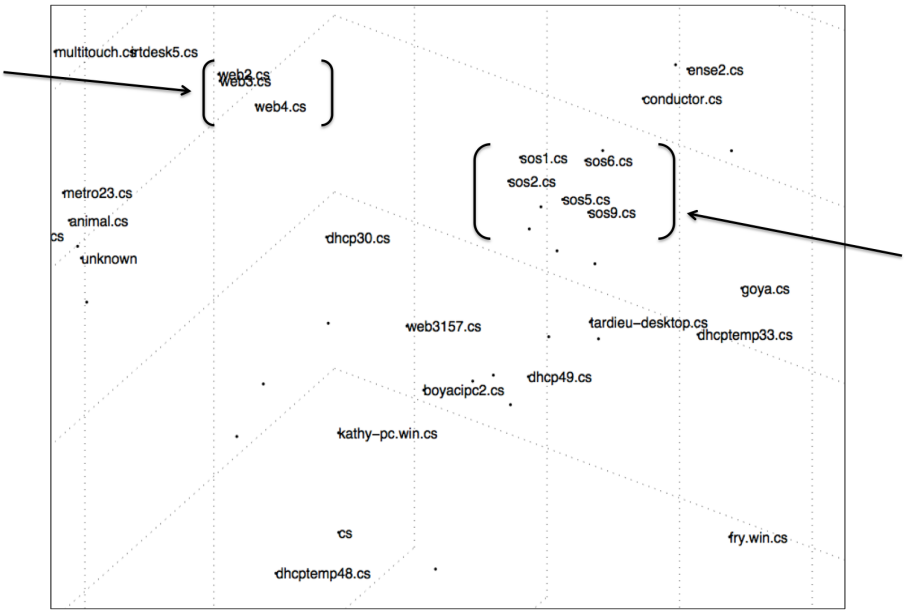


Figure 5.7: Kernel-PCA embeddings for CUCS dataset using PTM models.

cluster in the first and two clusters of behavior ("web" and "sos") in the second.

Utility in network research: Sub-space projection methods such as Kernel PCA is useful for situations where we wish to exchange data anonymously between organizations without sending the actual traffic instances. Instead, we send sub-space vector representations of behavior. This is can be done by projecting the data into the kernel PCA components. Similarity amongst hosts between different organizations, and similarity to other representative behaviors (attack traffic, vulnerable hosts, botnets members, *etc*), can be compared over the vector space embeddings this way in a manner analogous to the way hash-collisions are used in security. And If the comparison can be made to hash functions, then the set of "support" models used in the kernel computation $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_M\}$ can be thought of as the key.

5.5.2 Lower dimensional embedding with the Hellinger divergence

Non projection-oriented methods that focuses on producing better visualization results are also available. The field of low-dimensional embedding for visualization and data compres-

sion spans a range of different techniques. These include Multi-dimensional Scaling [Kruskal and Wish, 1978], Maximum Variance Unfolding [Weinberger and Saul, 2006], Minimum Volume Embedding [Shaw and Jebara, 2007] and other related methods [Roweis and Saul", 2000]. In our research context, low-dimensional embedding allows one 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, giving us a look at, what is essentially, the network behavior manifold.

These algorithms typically require the computation of a dissimilarity matrix, which is a matrix of pair-wise distances. This is a straightforward extension of our previously discussed algorithms. Note that the Bhattacharyya affinity calculated in probability product kernel is a positive definite metric, and not a distance metric, further it does not hold triangle inequality. However, the previously mentioned Hellinger-divergence does hold this property. This divergence is considered as a symmetric version of the well-known Kullback-Leibler (KL) divergence. Given that the kernel affinity values are not normalized by default, we can use the normalized Laplacian matrix to calculate the divergence matrix.

Low-dimensional embedding for network traffic:

1. Fit a PTM 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 matrix: $L = D^{-1/2} A D^{-1/2}$.
4. Compute Hellinger divergence matrix: $H[i,j] = \sqrt{2 - 2L[i,j]}, \quad \forall i, j = 1, ..., N$.
5. Use matrix H as input into any existing embedding algorithm such as MDS.

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.

Visualization of behavior changes across time: 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. The reader is encouraged to view a video demonstration on behavior tracking across time that is available on our website `http://www.cs.columbia.edu/~yingbo/MVE-DHS-2.mov`. This was produced using the minimum volume embedding technique described in [Shaw and Jebara, 2007]. The behavior of hosts on the CUCS network was split into 288 five-minute time-slices throughout the day, and their traffic embedding was computed at each of these time-slices. Plotting these embeddings provides an interesting view of how network behaviors can converge and diverge across the span of a single day. Clusters of behavior emerge and disperse and the overall shape of the distribution changes over this time period. Figures (5.8) through (5.13) show segments from this video.

5.6 Kernel machines and density estimation

Among all the machine-learning techniques used in practice by other disciplines, one of the most well known algorithms is the support vector machine (SVM). The discriminative power of SVMs, along with the simplicity of the classifier-implementation, makes this algorithm omnipresent in many toolboxes. This section discusses how to construct a support vector machine-based classifier for network traffic using the PTM with a probability product kernel. We demonstrate the strong performance of this classifier on actual network traffic.

5.6.1 Protocol-transition support vector machines

Fundamentally, the SVM is simply a linear threshold function in the Hilbert space defined by the kernel metric. Unlike optimization procedures such as EM, that optimize the parameter estimation based on the empirical risk (a function of the number of mis-labelings on the dataset), the SVM is optimized according to principle of Structural-Risk Minimization, which minimizes both the empirical risk and the structural risk of the classifier – which is a calculated as a function the classifier's VC-dimension. Additional discussion on structural risk minimization is not relevant for the purpose of this thesis. It is sufficient to state that

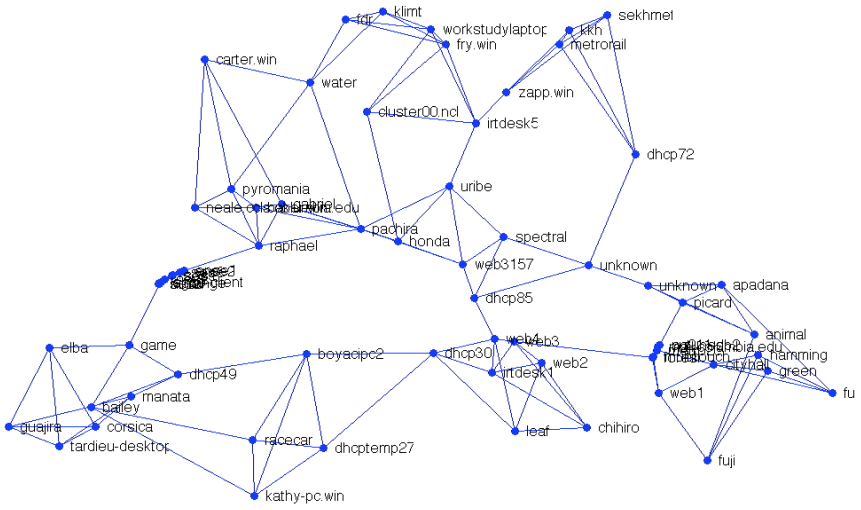


Figure 5.8: 4-D Behavior embedding. Time step 1.

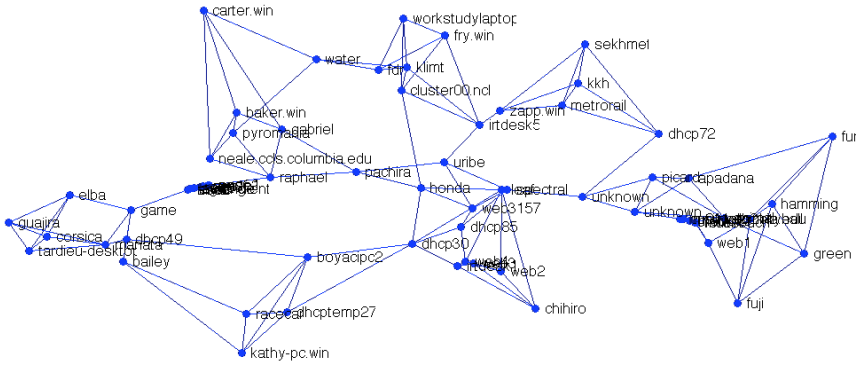


Figure 5.9: 4-D Behavior embedding. Time step 2.

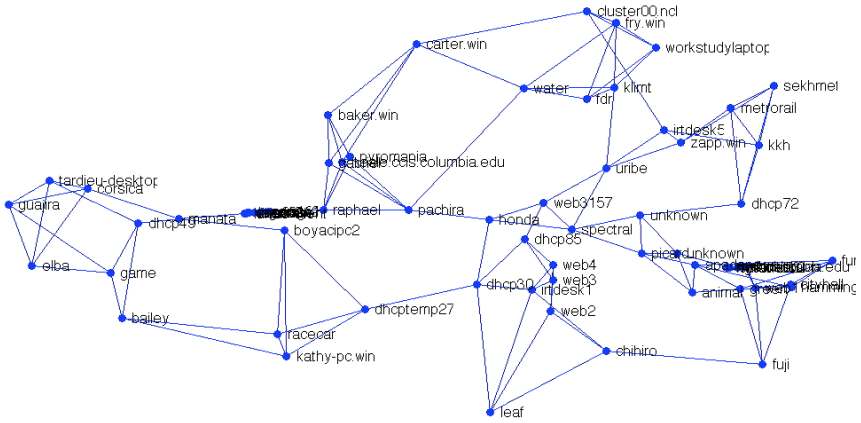


Figure 5.10: 4-D Behavior embedding. Time step 3.

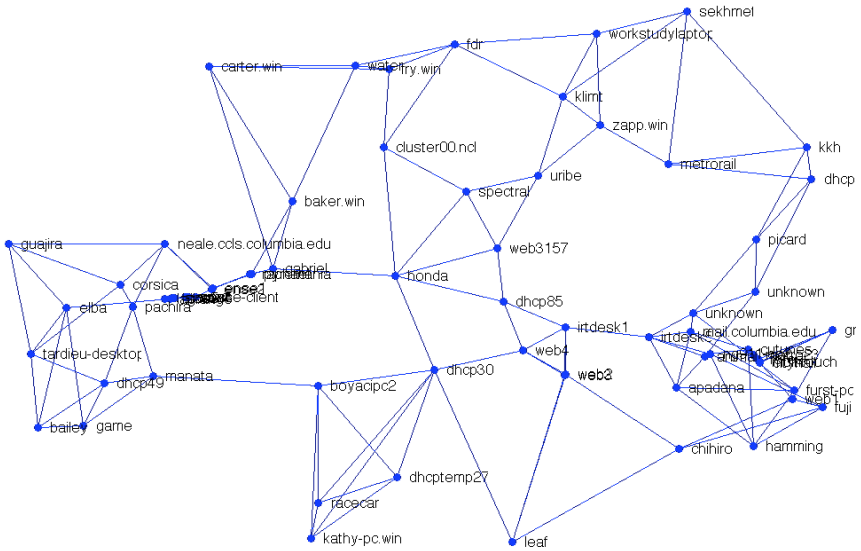


Figure 5.11: 4-D Behavior embedding. Time step 4.

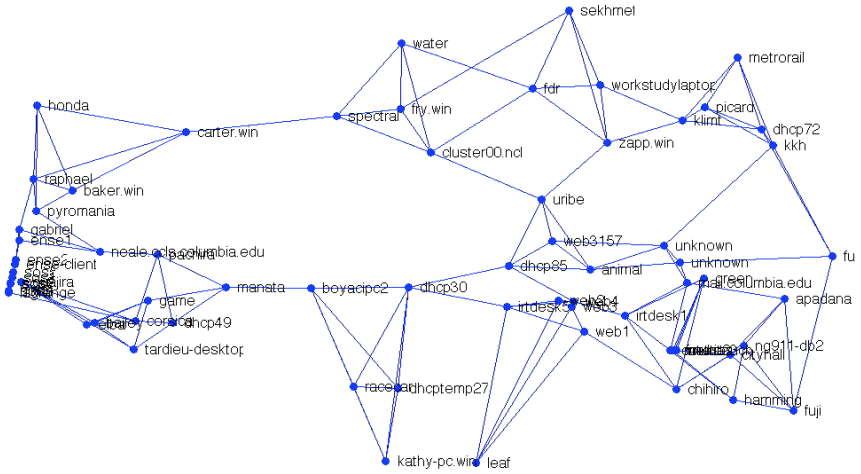


Figure 5.12: 4-D Behavior embedding. Time step 5.

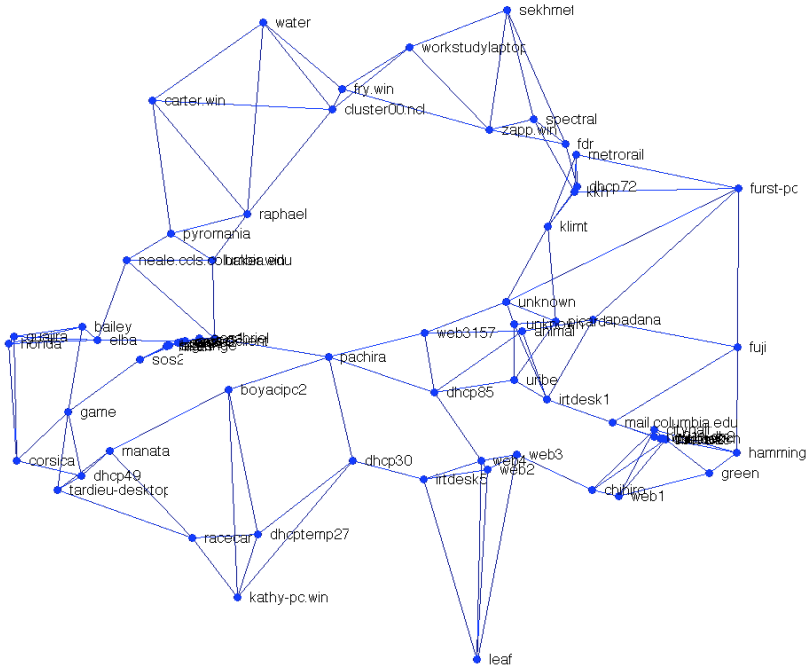


Figure 5.13: 4-D Behavior embedding. Time step 6.

derivation of the support vector machine for network traffic using the protocol-transition Markov model and the probability product kernel does not require much additional effort.

$$F(\mathbf{x}|\Theta) = \text{sign}\left(\sum_{i=1}^{M} \alpha_i y_i \mathcal{K}(\theta, \theta_i) - b\right). \tag{5.16}$$

Equation (5.16) shows the classifier function for the SVM. Optimization these parameters include estimating the set of support vectors needed, which takes the form of an index on the training data, and the set of weights (α_i) on these "support vectors" (although, in our case they are better referred to as support PTMs). These parameters are typically estimated by maximizing the following dual of the Lagrangian function:

$$L(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathcal{K}(\theta_i, \theta_j). \tag{5.17}$$

Maximizing the above in $\alpha_i \geq 0$ following the constraint $\sum_{i=1}^{N} \alpha_i y_i = 0$ can be done using quadratic programming. Here the data points being compared are the trained PTM model parameters $\theta_i, ..., \theta_N$, and the kernel function is the probability product kernel.

Table (5.5) shows results form between-class classification experiments. All false-positive rates were 0%, therefore the classification accuracy was displayed without ROC plot. These results are representative of between-class classification performance using a support vector machine with the PTM probability product kernel. It should be noted that training SVMs on network traffic datasets in this manner should be undertaken with care. Given that extremely large sample space of all possible time-series samples, and the fact that this classifier uses a subset of training samples as the classifier, the possibility of over-fitting to the small subset of training data used is more significant than in vector-space settings. Careful tuning should be used in practice to ensure that the parameters are estimated properly for the dataset and that the data used to train the classifier is representative of the distribution. In contrast, table (5.6) shows results form within-class classification experiments. The total number of correct and mis-predictions were used to calculate the accuracy as opposed to sweeping the margin results for a ROC curve. The "webX" machines represent the load balancer machines behind the Columbia Computer Science department's web-server. Given the overlapping nature of the traffic that these machines typically see,

Dataset	Classification Rate	False Positive Rate
'dhcptemp27' vs 'ng911-db2'	100%	0%
'web4' vs 'boyacipc2'	100%	0%
'chihiro' vs 'neale'	100%	0%
'cluster00.ncl' vs 'cityhall'	100%	0%
'corsica' vs 'fdr'	100%	0%
'ng911-db2' vs 'uribe'	100%	0%
'dhcp85' vs 'furst-pc'	100%	0%
'neale' vs 'spectral'	100%	0%
'cityhall' vs 'gabriel'	100%	0%
'raphael' vs 'baker'	100%	0%
'cutunes' vs 'zapp'	100%	0%
'dhcp49' vs 'animal'	100%	0%
'leaf' vs 'hamming'	100%	0%
'multitouch' vs 'mail'	100%	0%
'cityhall' vs 'boyacipc2'	100%	0%

Table 5.5: PT-SVM classification accuracy for inter-class behavior; all classes were linearly separable. Representative of between-class classification results.

DATASET	'WEB1'	'WEB2'	'WEB3'	'WEB4'
'WEB1'	—	100%	100%	100%
'WEB2'	—	—	57.5%	87.5%
'WEB3'	—	—	—	85%
'WEB4'	—	—	—	—

Table 5.6: PT-SVM classification accuracy for intra-class behavior. Web[1,2,3,4] are NAT'd hosts supporting a load-balanced web-server.

lower performance is to be expected.

5.6.2 Non-parametric behavior modeling via kernel density estimation

Having discussed discriminate learning with the PTM support vector machine. This chapter concludes with a discussion on generative learning in the non-parametric case using kernel density estimation (KDE). KDE, also referred to as the Parzen-window method, is a method of approximating the parameters of non-linear distributions, as a function of similarity over a set of training points. In some ways, this is a generative analogue of the SVM function. Similar to the previous kernel algorithms, KDE requires only a valid kernel metric between samples, and the density estimate may be solve as follows:

$$F(\theta;\Theta) = \frac{1}{N}\mathcal{K}(\theta,\theta_j). \tag{5.18}$$

Where $\Theta = \{\theta_1, ..., \theta_N\}$ represents the set of learned PTM model parameters. Or a simple non-parametric likelihood estimator for any instance of a time-series data $\mathbf{x}$ can be recovered as the mean of the likelihood evaluations over PTMs trained over the entire dataset:

$$F(\mathbf{x};\Theta) = \frac{1}{N}\sum_{i=1}^{N} p(\mathbf{x},\theta_i). \tag{5.19}$$

The KDE method is used for anomaly detection later in this thesis to detect anomalous traffic from the network, as well as to show how our anonymization preserves the statistical

distribution by detecting anomalous clusters of traffic post-anonymization and demonstrating that original anomalies are present. These results are presented in § 7.5.2.

Concluding remarks

This section described our kernel-based framework and the class of algorithms that our work naturally extends into. We described the probability product kernel, and how to use this method, along with the PTM, to build a graph representation for the distribution of host behaviors on a network, and how a graph-partition-based approach can be an intuitive method with which to segment a network into distinct classes of behavior. We also showed how to guarantee a minimum anonymity set size using the recursive partition function.

The PPK is one of many different kernels in machine learning. It is also possible to combine results from several kernels to incorporate multiple measurements of similarity to improve performance. This field include techniques related to multiple-kernel learning [Gönen and Alpaydin, 2011]. One potentially useful feature set is incorporating features from the topology of the network (cf. § 8.2).

Chapter 6

Network Traffic Synthesis and Simulation

Introduction

Data-synthesis is an important, yet under-explored, aspect of network-trace anonymization. While sampling statistical elements within a table according to a privacy-preserving methods in the context of differential privacy has been studied for microdata, generating network traffic to mimic behavior for the purposes of obscuring source-identity in network capture datasets has not received equal attention. Traffic synthesis has been studied in works related to cover-traffic generation in overlay and anonymous VPN implementations, as well as tangentially-related fields in network measurement such as generating traffic for load-balance testing, routing testing, and for evaluation of intrusion detection systems. The intuition behind using synthetic data to mask source-identity is nevertheless understood. In this chapter we demonstrate how to generate such data for network-trace anonymization through sampling from the PTM model and the use of a custom packet-synthesis library which we have designed.

This chapter is outline as follows: we describe the theory and implementation behind netflow-layer statistical data synthesis in § 6.1. This section discusses how synthetic data is sampled form the protocol-transition Markov model, and answers the question of how to evaluate the fidelity of the synthetic data. In § 6.2, we present algorithms for behavior shaping and behavior interpolation when given input of different sets behavior profiles. We further show how synthesize can be used to perform behavior regression, where we can essentially fill-in missing gaps of traffic, how to synthesize behavior across time in order to faithfully mimic activity throughout the day. Finally, we describe how to generate sessionizeable packet streams using these netflow-layer features. Results within this chapter were published in [Song *et al.*, 2011].

6.1 Synthesizing flow-layer statistical samples

Synthesizing traffic for anonymization has been studied primarily within two distinct contexts. The first is the use of cover-traffic in online anonymization systems, such as mixnets or onion-routing networks like Tor. These networks provide source-anonymization in a continuous manner, depending on the connectivity patterns in the users. Due to the nature

of online source-anonymization, these systems are sometimes vulnerable to timing-analysis attacks. An extreme example is when only one member is active on a mixnet at a given time, an attacker observing the exit traffic can easily determine the identity of the source. Cover-traffic is then used to provide an additional layer of obfuscation that raises the level of difficulty for reattribution [Troncoso and Danezis, 2009; Freedman and Morris, 2002; Edman and Yener, 2009]. The second is the context where this type of algorithm is used is in statistical database anonymization. In the microdata context, replacing elements of a table with samples from a distribution in a privacy-preserving scheme has been explored in techniques such as differential privacy [Dwork, 2006]. Traffic shaping is a related technique that is also designed for obscuring signatures in order to prevent attribution. This often takes the form of traffic padding to prevent statistical analysis. It has been discussed in mixnet implementations [Dingledine *et al.*, 2004] and analyzed in more detail [Syverson *et al.*, 2001]. Results have been achieved for anonymization of specific protocols and applications. An example of such protocol obfuscation is the traffic morphing techniques based on packet-size manipulation studied in [Wright *et al.*, 2009].

This chapter discusses techniques for synthesizing traffic for network packet capture datasets. This is similar to the online setting, however we have the benefit of having all of the available traffic for a host that we wish to mimic. This allows accurate estimates for behavior and samples can be generated such that it is quantifiably similar to the original. Rather than manipulating the size and distribution of traffic in order to add noise to the signal, we can actually re-shape traffic based on specific targeted behavior recognizable by our models.

Traffic synthesis contains two components: the first is sampling statistical records from our time-series model for particular host that we wish to mimic. Or sampling from a model trained to encapsulate a specific type of behavior (web server, vs. database server, etc). The second component consists of translating thees statistical features into actual packets that are fully sessionizeable and consistent with TCP/IP specifications. The first problem is a matter of statistical sampling from a time-series distribution, and the second is mostly a technical matter of conforming to RFC specifications.

6.1.1 Sampling from the PTM

Since the PTM is a time-series model with observable states, synthesizing a new Netflow sequence translates to drawing a statistical sample from this model. This includes drawing an initial state, outputting the port value associated with that state, drawing from the emissions distribution associated with that state, drawing the most-probable subsequent state according to the transition distribution, and repeating this process; algorithmically, these steps involve sampling from three different distributions. A few notations need to be introduced before describing the algorithm: 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 $\mathbf{p} = \{p_1, p_2, \ldots, p_k\}$. Assuming an alphabet of k characters with 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 the synthesis algorithm is given as follows:

SYNTHESIZE-NETFLOW(θ, n)

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

SYNTHESIZE-STATS(θ, n)

1 ▷ Use the frequency estimate to set the initial state
2 $d_0 \sim \mathcal{M}(\pi_1, \pi, \ldots, \pi_k)$
3 ▷ Src/Dst port model used to estimate the pairing
4 $s_0 \sim \mathcal{M}(b_{d_0,1}, b_{d_0,2}, \ldots)$
5 $x_0 \sim \mathcal{N}(\theta_{s_0})$ ▷ Sampling the volume
6 ▷ The rest is generated by induction
7 **for** $t \leftarrow 1$ **to** $n - 1$:
8 **do** $d_t \sim \mathcal{M}(a_{d_{t-1},1}, a_{d_{t-1},2}, \ldots, a_{d_{t-1},k})$
9 $s_t \sim \mathcal{M}(b_{d_t,1}, b_{d_t,2}, \ldots)$
10 $x_i \sim \mathcal{N}(\theta_{s_t})$
11 **return** $[\mathbf{s}, \mathbf{d}, \mathbf{x}]$

Since the states within the PTM are observable, the associated port values are unambiguous. The transition probability is represented by table $\mathbf{T}[a, b]$ that indexes the likelihood of transition between any pair of states. For a table $\mathbf{T}[a, b]$, the row-index a represents the current state while the column index b represent the subsequent state and the value within that table, $\mathbf{T}[a, b] = p(s_t = b|s_{t-1} = a)$, denotes the probability of the transition from a to b. Further, the table is row-normalized: $\sum_i \mathbf{T}[a, i] = 1$. Drawing the most-likely subsequent state, $t + 1$, is equivalent to taking a maximum-likelihood sample from a multinomial distribution with parameters defined by these individual rows of the table.

Recall that the states correspond to destination port values, and is thus only one-half of the sample space. The PTM also incorporates multinomial distributions for the associated source-port values, which are tracked in the $\mathbf{b}$ variables. These track the conditional probability of observing any particular source port, given a destination port. A conditional probability is used instead of the joint probability in order to prevent under-fitting problems, as the space of source and destination port pairings is typically exponentially larger than the space of available training samples. Source and port destination pairs are driven by network protocol design and not uniformly random. That is, modeling $p(s_i|d_i)$ is sufficient as opposed to modeling $p(s_i, d_i)$. Second order modeling for source and destination pairings is similarly redundant.

Finally volume information (packet count) x_i is drawn from the Gaussian mixture associated with each unique state i. Function SYNTHESIZE-STATS shows the steps of the synthesis algorithm. Recall that we utilize port-histograms in our feature representation, in order to reduce the total possible number of states. This bin value is the output of the synthesis algorithm and must then be converted back to the normal data-space by the following function.

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

1 $[\mathbf{s}', \mathbf{d}', \mathbf{x}'] \leftarrow \emptyset$
2 $n \leftarrow$ LENGTH($\mathbf{s}$)
3 **for** $t \leftarrow 1$ **to** n :
4 **do** $s'_t \leftarrow$ BIN-TO-PORTS(s_t)
5 $d'_t \leftarrow$ BIN-TO-PORTS(d_t)
6 $x'_t \leftarrow \exp(x_t)$
7 **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)

1 **if** $x < 1026$: ▷ Well known ports
2 **then return** x
3 **elseif** $x > (1025 + h)$: ▷ Ephemeral ports
4 **then return** RAND$(1, \ldots, 16384) + 49151$
5 $z \leftarrow (49152 - 1025)/h$ ▷ Registered ports
6 **return** $1025 + (x - 1025) \times h +$ RAND$(1, \ldots, h)$

For each port histogram bin, we sample the port value represented by the range of that bin according to a uniform distribution. It is possible to further use a non-parametric model to represent the global port-distribution for each bin. If this is used then all of the data within the network may be used to train these models, in order to avoid under-fitting. Port selection in TCP is conditioned on the implementation and internal condition of the operating system, and not a function of user behavior, therefore training data from across all behavior classes may be used to train these models. However, such granularity is often unnecessary; more accurate simulation of port pairings may be achieved by using a smaller bin size in the port histogram. This can be adjusted on a per-network basis in order to find the optimal performance with respect to synthesis and model accuracy.

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 6.1: Log-likelihood evaluation of genuine, synthetic, and randomly-sampled data across datasets (average of 10 trials.)

6.1.2 Quantifying similarity and data fidelity

One of the primary questions pertaining to data synthesis is how to evaluate the fidelity of the synthesized data. If the data is meant to be used for anonymization, what is the measure by which we can be sure that one sample is more useful than another. Given that our techniques are statistics-driven, the answer to this question is rather obvious: the measurement quality is simply the likelihood evaluation of the synthesized dataset using the behavior models. Let $\mathbf{x}_i^* \sim p_{PTM}(\theta_i)$ represent a sample drawn from the PTM trained for host i. We evaluate the fidelity of the sample with $p(\mathbf{x}_i^*|\theta_i)$. A synthetic sample consistent with the behavior of the original model would yield a higher log-likelihood score than a poorly recreated sequence.

In practice $\log(p(\mathbf{s}, \mathbf{x}|\Theta))$ is used to avoid underflow errors. Table (6.1) shows likelihood comparisons for synthesized data over the different datasets. We compare the log-likelihood scores of synthetic data with the original (genuine) data as well as data piece together by randomly selecting subsets traffic belonging to other hosts. In this latter case, represented in the table by "Random sampled," a naive way of data synthesis is simulated, where we randomly piece together segments of traffic belonging to other hosts to synthesize traffic for a particular host. This method gives us a more challenging performance baseline to compare against, as opposed to synthesizing random data via sampling from a uniform distribution, given that we are using segments of actual traffic. Random data sampled in that manner yields log-likelihood scores that are orders of magnitude lower and are thus not

interesting for comparison. All results are average of 10 randomized trials. As expected, our synthesized data consistently exhibit higher log-likelihood scores than the randomly sampled data, but lower than the genuine data, which would represents an upper bound. The results confirm how we would expect data synthesis algorithms to perform, that data synthesized from a behavior model is more realistic than randomly sampled data, but less than genuine data.

6.1.3 Anonymization via. model-exchange

Traffic synthesis can be used not only to obscure the source signal within a dataset but also can, in theory, replace the original traffic, given accurate enough models and simulation. If the models can achieve 100% accuracy on the training dataset, for example, and if the synthesis algorithms could consistently generate synthetic data with respect to these models. Then it may be possible to simply exchange model parameters between organizations for research purposes rather than exchanging anonymized data. For certain research problems where only independent measurement of behavior is necessary, such as detection of intrusions, measurements on the distribution of services, and other volumetric measurement, using these models would be the next best thing to having authentic data. The benefit of exchanging only models is that a high level of anonymity may be achieved given that the potential for side-channel information disclosure is minimized. Using the synthesis techniques described in this chapter, the receiving party can re-synthesize the dataset according to these models, if necessary. This aspect of research is part of our future work and will be discussed in further detail in that chapter.

6.2 Behavior-based traffic interpolation

Behavior interpolation can be achieved in the simplest way by using a switched PTM. A switched time-series model is simply one where multiple models are joined together and an independent variable is used to track which model is active at a given time. The probability of "switching on" any specific model can either be a multinomial distribution if the transitions are independent, or can be modeled explicitly using a transition probability, much in

the same way as the state-transitions within each model operate. This generates traffic that represents an interpolation of behavior between the different behavior models, and would be reflected in the model parameters trained over this synthesized data; a hypothesis that will be experimentally confirmed in this section. A user-specified distribution on the switching variable can be used to control the degree of interpolation.

SYNTHESIZE-MIXED-NETFLOW provides the pseudocode for this procedure:

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

1 $[\mathbf{s}, \mathbf{d}, \mathbf{x}] \leftarrow \emptyset$
2 **for** $t \leftarrow 1$ **to** $\lfloor n/c \rfloor$:
3 **do** $i \sim \mathcal{M}(\mathbf{p})$
4 $[\mathbf{s}_t, \mathbf{d}_t, \mathbf{x}_t] \leftarrow$ SYNTHESIZE-NETFLOW(θ_i, c)
5 $[\mathbf{s}, \mathbf{d}, \mathbf{x}] \leftarrow$ APPEND($[\mathbf{s}_t, \mathbf{d}_t, \mathbf{x}_t]$)
6 $t \leftarrow t + 1$
7 **return** $[\mathbf{s}, \mathbf{d}, \mathbf{x}]$

In the most straight-forward implementation, 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. Given that, in practice, interpolating behavior would often be done on similarly matched hosts, the statistical divergence within these traffics blocks would be minimal. 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 more similar to each other and contain a high degree of overlap among port values, in which case it would allow the models to transition between each other more smoothly. In practice, c can be adjusted based on the model similarities.

A visual similarity-measurement technique to quantify the the fidelity of the synthesized data in this interpolation setting. We confirm our hypothesis on the interpolation effect by

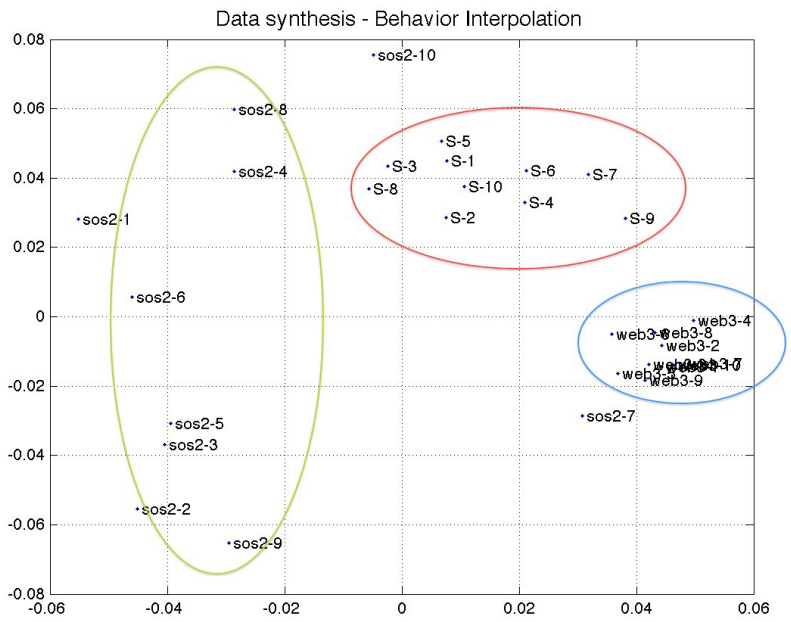


Figure 6.1: Behavior interpolation: Two clusters of behavior profiles are embedded ("sos2" and "web3") along with a new class of behavior ("S") that interpolates between these two models.

using the embedding techniques described in the kernel chapter to visualize the distribution of the original and synthetic points with respect to each other. In this experiment, 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.

Figure (6.1) shows the embedded results of interpolation two distinct classes of behavior. Visually, we can confirm that the synthesized data indeed falls between these two distinct classes of behavior. In this experiment, we trained PTMs on traffic belonging to two hosts: "sos2" and "web3." These models were the used in the interpolation-synthesis algorithm described above in order to generate a set of synthetic data which is represented in the embedding as class "S." The samples from sos2, web3 and S were then embedded into two dimensions using multi-dimensional scaling with a PPK-calculated Hellinger divergence matrix. The embeddings are distributed according to what we would expect. The two

classes are clustered together, and the third class is embedded between these two. Note that the distance relationship is not exactly linear in this space – the linearity is actually in the high dimensional Hilbert space imaged by the probability product kernel, but appears non-linear in the lower dimensions.

6.3 Behavior-based traffic regression

Often network traffic is not captured continuously and gaps in traffic may occur, or traffic instances may be unaligned. One interesting problem is how can we fill in this missing gap. We refer to this problem as traffic regression. 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 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 samples and pick the one that yields the highest likelihood given the choice of probability model $p(\mathbf{s}, \mathbf{x}|\Theta)$, or more efficiently implemented as generating a very long sample and using a sliding window to calculate the block of traffic that yields the highest likelihood estimate with respect to the boundary traffic instances.

This process takes the following form:

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

1 $n \leftarrow$ SOME LARGE NUMBER

2 $D \leftarrow [\infty, \infty, \ldots, \infty]$ of length n

3 $[\mathbf{s}', \mathbf{d}', \mathbf{x}'] \leftarrow$ SYNTHESIZE-NETFLOW(θ, n)

4 **for** $t \leftarrow 4$ **to** $n - c - 4$:

5 **do** $D_t = \sum_{i=t-3}^{t} ||d_i - d'_i|| + \sum_{i=t+c}^{t+c+3} ||d_i - d'_i|| \ldots$

6 $+ \sum_{i=t-3}^{t} ||s_i - s'_i|| + \sum_{i=t+c}^{t+c+3} ||s_i - s'_i||$

7 $j \leftarrow$ FIND-INDEX-OF(MIN(D))

8 $[\mathbf{s}*, \mathbf{d}*, \mathbf{x}*] \leftarrow [s'_{j,\ldots,j+c-1}, d'_{j,\ldots,j+c-1}, x'_{j,\ldots,j+c-1}]$

9 **return** $[\mathbf{s}*, \mathbf{d}*, \mathbf{x}*]$

This algorithm synthesizes a single long sequence, and a sliding window of fixed size is swept across the generated data to find the indices where the boundaries are most similarly aligned between the original and the synthetic data. The method of similarity measurement is simply the 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

Figure 6.2: Filling in missing data: (Left) shaded entries represents missing data. (Right) synthesized entries.

Figure (7.12) shows an example of this regression. In this experiment, we trained a model for a particular host and took an unseen segment of traffic from the same host for testing. From this testing sample, we removed a portion of the traffic – 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. Statistically, this is the maximum likelihood estimate. However, it is easier to visually appreciate the similarity in the real and synthetic samples.

6.4 Passive network-device fingerprints

The next section provides a description for how to synthesize packet-data that is consistent with the netflow-layer statistical representation. Before that topic is covered, however, a brief mention of a related topic of passive device fingerprints is necessary. In addition to statisical behavior-based fingerprints and overt identifiers such as IP or Ethernet addresses, less-obvious content-based signatures also exist within the TCP packet headers, in the form of predictable values in certain header fields. These potentially discriminative features include TTL values and Window sizes set within TCP SYN packets. These values are determined, not by application level host behavior, but rather by the implementation of the operating system, and the implementation of the network protocol stack. Given that TCP is agnostic to layer-2 implementation not to mention host OS implementation, such features such as maximum segment size, TCP window size and path MTU must be negotiated in a handshaking procedure during the initialization of the TCP session between two different hosts. During this initialization the hosts submit their own acceptable values to each other, and it is this exchange of information that present the avenue for passive OS fingerprinting. These signatures are typically used to identify the operating system of the connecting hosts, are referred to as "passive" fingerprints to distinguish them from those fingerprints retrieved by actively probing of the host. `p0f` [Zalewski, 2006] is a well-known instance of the passive-OS fingerprinting tool that uses characteristics of the SYN packet, primarily, to perform passive OS identification.

Active OS detection leverages the fact that different operating systems, and different versions of the same operating system (Linux 2.4,2.6...) have different implementations that will behave differently when handling malformed packets. `nmap` is a prominent example of active OS fingerprinting tool. Other forms of active fingerprints also exist, such as the appearance of port-scans, DDoS attacks, and other behaviorally recognizable artifacts within network traffic that can distinguish one particular host from another, or a particular dataset from another. This topic is discussed later, in the chapter on network trace anonymization.

Passive OS fingerprints are relevant in the context of synthesis in that the synthesized packet streams must be consistent with these features in addition to the higher level statistical representation. The next section will describe how this is achievable.

6.5 Synthesizing TCP-layer packet traffic

This section describes the technical details in crafting sessionizeable packet streams that are consistent with the statistical samples synthesized using the techniques of the previous section. Recall that behavior features that we study take the following form:

$$\mathbf{x} \in \{timestamp, srcaddr, srcport, dstaddr, dstport, pktcount\}.$$

In our setting, each of these records represent the unidirectional flow of information from one host to another and a collection of these instances represent a session. A collection of sessions then represent the traffic set of a particular host. The prior section described how to synthesize each of the fields within this record, with the exception of the timestamp, which we will discussed in a subsequent section. The IP address will be obfuscated values in the resulting dataset, so they may be chosen at random or in a prefix-structured manner if network topology and subnet distribution is to be preserved. Given this set of data, implementing the packet transform is a mostly a technical challenge, and not a conceptual one. The difficultly lies primarily in ensuring the packets are consistent with RFC 1122 specifications on the implementation of the TCP/IP layers [Force, 1989].

We have written TCP-session synthesis library in Python for assist in this last step of the transformation. This library allows one to program the synthesis of packet streams using the statistical entries described above. This library is essentially a simplified software stack for the TCP layer. The main challenge for us to consider is producing sessionizeable packets consistent with the given statistical behavior. The underlying characteristics of the IP-layer is, for the most part, independent of the application behavior. Therefore, we built our library on top of the Scapy PCAP programming library [Biondi, 2012] and use their handling of the IP stack to transparently craft the individual IP packets.

The library transparently handles the establishment of the TCP session and ensures the transmissions characteristics are consistent with the statistical information. Given that TCP is a full-duplex connection, the corresponding sequence numbers and window-size values for each session are tracked accordingly. These steps are as follows:

1. Establish the TCP handshake by crafting the TCP SYN packet for the originating host, consistent with the input OS fingerprint. The receiver host will often also have

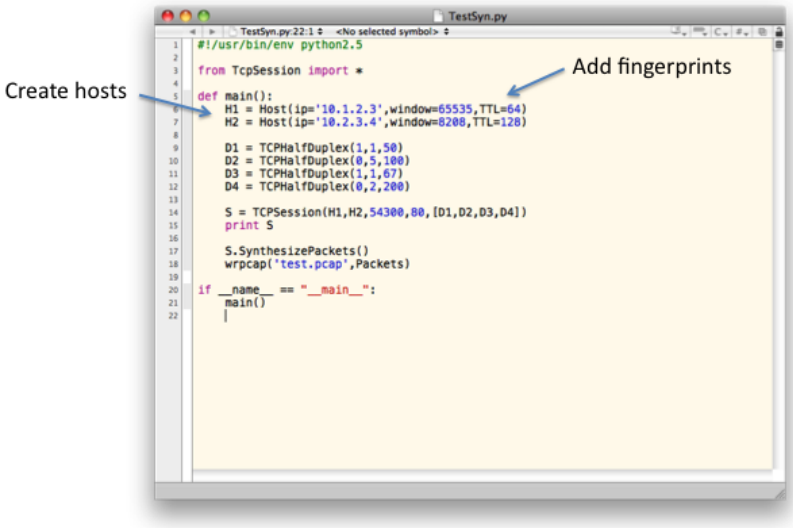


Figure 6.3: Programming with the Flow-to-PCAP TCP-session-synthesis library.

passive fingerprint, and this is incorporated into the SYN+ACK response.

2. Control variables such as window sizes are established given the host pairings.

3. Initial values for the TCP sequence number are generated at random.

4. Packets are generated for time entry t according to the statistical samples $\mathbf{x}_t$. The TCP sequence numbers are updated. Random bytes are used to fill the payload portion of each packet. The packet sizes are kept consistent with the TCP sequence numbers and the path MTU. We pad the packets so that the maximum allowable amount of information is used, according to the MTU (c.f. figure (4.2) for packet size distributions).

5. After the session is complete the FIN, FIN+ACK packets are generated.

Scapy is used to automatically maintain consistency in the IP header values with respect to checksums, flags, and the packet IP ID values.

Figure (6.3) shows a sample program written to craft a TCP session using our library. Figure 6.4 shows that these synthesized packets passes all checks in Wireshark and are fully sessionizeable.

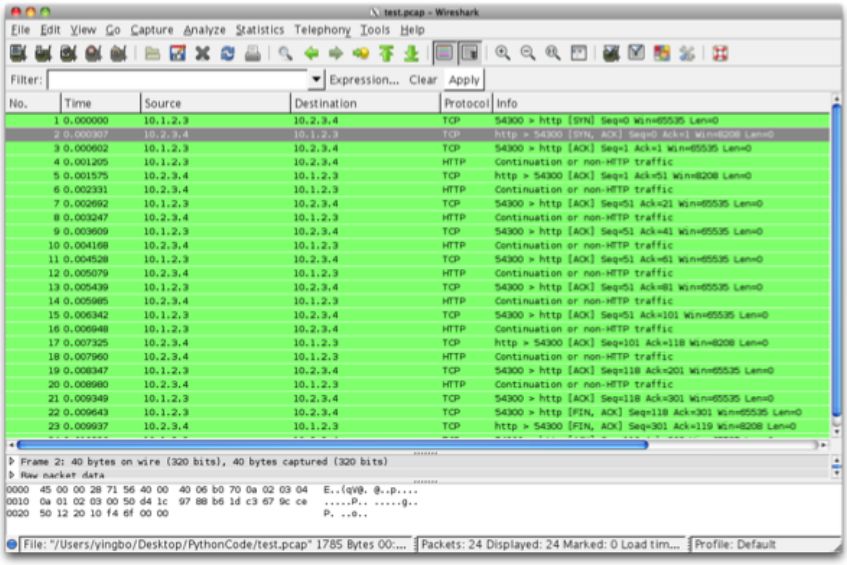


Figure 6.4: Synthesized packets are sessionizeable and passes all checks in Wireshark.

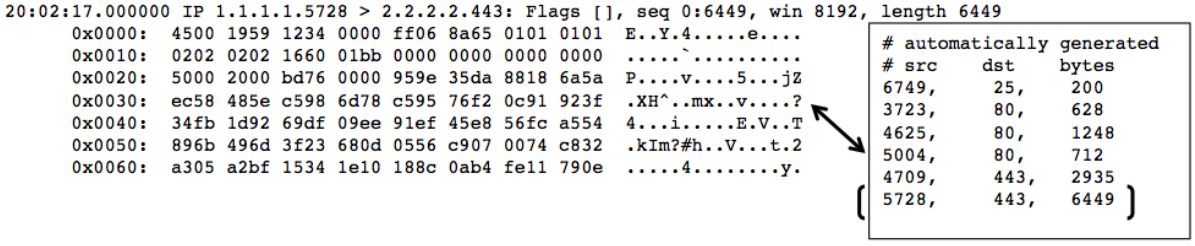


Figure 6.5: Translation between statistical representation to PCAP.

In practice, the programming API would read statistical features from the generated statistical entries and the user will not need to manually input these values. Figure (6.5) shows details of an instance of synthesized packets. The translation between statistical representation and packets is shown. One limitation of this method is that the generated traffic is fairly "clean" and network jitters are not currently reproduced. 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.

6.6 Time-driven traffic synthesis

The timestamp feature was the last missing element of the synthesize procedure. This is because the PTM model does not explicitly trace time and as such as no concept of exact traffic generation times or session inter-arrival times. Incorporating timing modeling can be done using mixtures of exponential distributions, and is discussed in the future works. In order to synthesize traffic conditioned on time, a separate distribution conditioned on time may be used. However, if a more accurate model is desired one such solution to this limitation is training independent PTM models in a piece-wise manner on segments of traffic conditioned on time. This is a rather simple engineering solution to a conceptually challenging problem.

One of the reasons why modeling traffic conditioned on time is that 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. It reflects the issues with the HMM performance described in the model sections. If a traffic profile is conditioned on a period of time during the day then each instance of training traffic requires measurement from a different day in order to be consistent.

We chose the first approach mentioned in adopting a separate distribution that models the amount of traffic a host generates on a given day. This model breaks a day into

288 individual five-minute time slices and, 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:

$$\mu_t = \frac{1}{m}\sum_i^m y_{i,t} \tag{6.1}$$

$$\sigma_t = \frac{1}{m}\sum_i^m (y_{i,t} - \mu)^2. \tag{6.2}$$

This implicitly assumes independency between the individual time slices. 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** $v_t < \epsilon$: ▷ 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 ▷ Merge all PCAP files into $F*$
8 $F* \leftarrow$ MERGE-PCAP$(F_1, F_2, \ldots, F_{288})$
9 **return**

In the above pseudo-code STATS-TO-PCAP is the function for generating packets from a statistical sample. In our experiments our models have yielded interesting results, such as replicating port-scanners found in the NSA dataset.

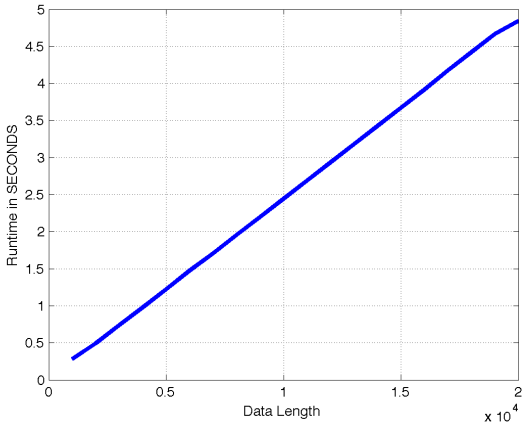


Figure 6.6: SYNTHESIZE-NETFLOW maintain $O(n)$ growth with respect to number of flows. Large traffic set with 20,000 flow entries took less than five seconds to synthesize.

6.7 Runtime evaluation

The memory footprint of our models is 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 or so network services (c.f. Table (4.2)), the storage requirement of 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.

Figure (6.6) shows the runtime of this synthesis algorithm. Given that our algorithm utilizes a single-pass over the dataset, its runtime growth is linear in $O(N)$. Even in SYNTHESIZE-DAY where two loop layers exist, the outer loop is upper bounded by a relatively small constant, therefore the entire system still grows in $O(N)$. Figure (6.6) shows

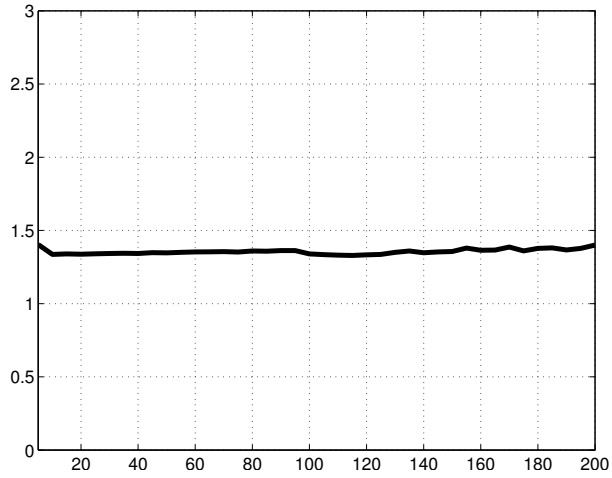


Figure 6.7: SYNTHESIZE-NETFLOW runtime is largely independent of the model size.

the runtimes for synthesizing the behavior of a host utilizing 16 network services. The results demonstrate that synthesizing a (relatively large) 20,000-element Netflow record required less than five seconds. Results were derived from computation 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 performances may be achieved.

Figure (6.7) shows that the runtime of the synthesis algorithm is independent of the model size. This is because the state-transitions is largely independent on the number of states. For each state transition an index a column within a matrix is selected the size of the state-space adds computational cost to the multinomial-distribution sampling function that selects each subsequent state, however this effect is minimal, and only adds a small multiplicative factor to the overall runtime.

Concluding remarks

In this chapter 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, as well as be used 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.

Chapter 7

Network Trace Anonymization

Introduction

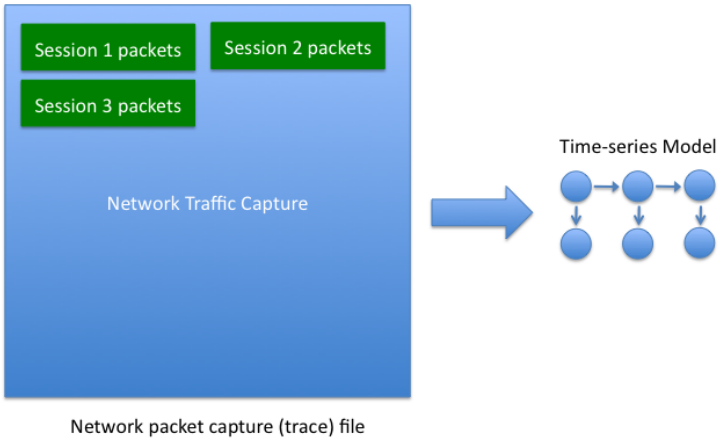
Having described the machine learning-based components of network host behavior modeling, similarity measurement, and traffic synthesis in the previous three chapters, this chapter shows how all of these techniques are combined to create a source-anonymization system consistent with the formalization presented in Chapter 3. This chapter further shows how the minimum anonymity set size is estimated and derive these measurements for various datasets. We show how to ensure that this minimum requirement is met while preserving the behavior distribution. The technical details behind the traffic-merging functionality are presented, such as obfuscation requirements for specific packet header values. Measurements on the statistical distributions before and after anonymization demonstrate the statistics-preservation quality of this transformation. The stability of the algorithm and the quality of the matched groupings are also evaluated. Algorithms for behavior level traffic shaping and methods by which we can measure these changes are also presented. Further, connections to existing anonymity metrics are made, such as k-anonymity, anonymity set size, and other metrics, in order to show how existing anonymization theory are relatable to our work and how these methods may be used to guide the use of this technology. Potential areas of information leakage are explored and methods to mitigate these problems are offered.

7.1 Anonymity via crowd of similarly-behaving peers

We have demonstrate in the previous chapters that statistical models for static and temporal traffic behavior may be used to identify groups of individuals with distinctively similar behavioral patterns. The accuracy of the models in classification experiments confirms that behavior is quantifiable, measurable, and that groups of similar behaviors can be identified using these methods. We further showed that traffic consistent with targeted behavior can be synthesized, and confirmed this with fidelity measurements that shows the statistical significance.

This sections walks through the conceptual overview of the anonymization process.

Figure (7.1) show the first step of the process where PCAP data, containing session



Step 1 - Feature Extraction

Figure 7.1: PTMs are trained for each host within a network trace file in steps 1 and 2.

information is extracted into flow-layer statistics and PTM models are trained for each host.

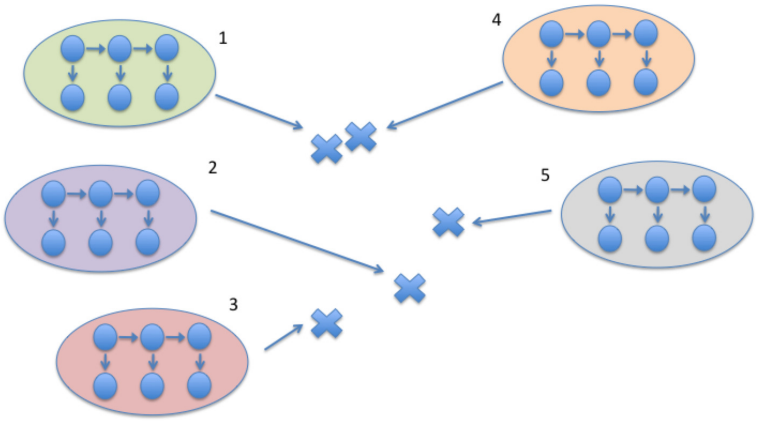
Given the model, we can conceptualize each host as a single point in an abstract behavior space defined by the PTM, shown in figure (7.2).

The probability kernel provides a pair-wise similarity distance metric for these behavior models and can be used to define a fully-connected graph, as shown in figure (7.3) where user behaviors are considered nodes and the edge weights are pair-wise probability product kernel affinity evaluations. Without the need to assume a parametric model for the overall host-behavior distribution (as in EM or KMeans), recovering the optimal partition is a function of obtaining the minimum weight graph-cut solution. A fast and efficient spectral approximation to the normalized ratio cut is used, and the optimal partition is recovered.

Given this partition the clustering in the original model space is identified (figure (7.4)).

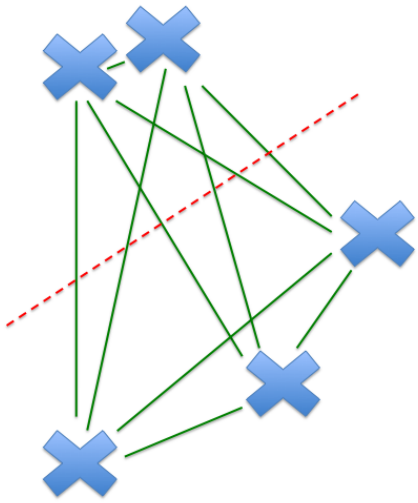
The groups are now merged and group identities are assigned.

Finally, this clustering is actuated in network data by modifying the packets such that the new sessions are consistent with the merged identities. Note that this matching does not need to use hosts from within the same dataset, rather multiple sources for network traffic may be clustered simultaneously, providing additional anonymity by enlarging the anonymity set size.



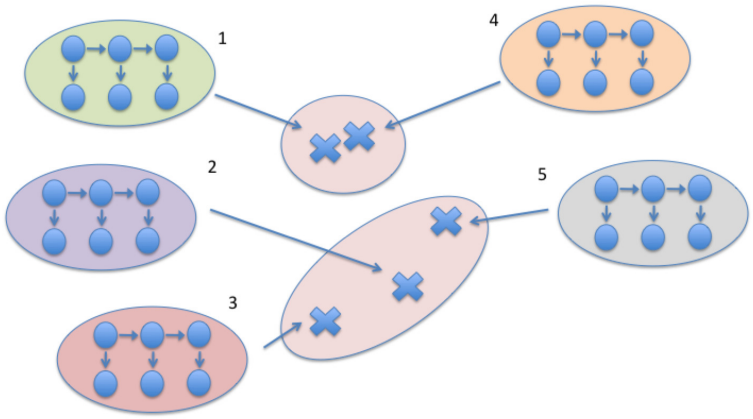
Step 2 - PTM-Space

Figure 7.2: Time-series models can be conceptualized as points in a time-series space.



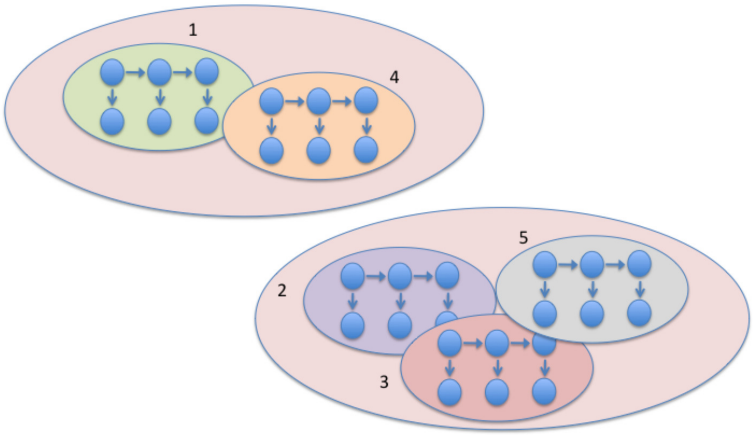
Step 3 - Kernel metric casts the behavior profiles as nodes within a fully-connection graph.

Figure 7.3: These behavior models are conceptualized as points within a graph where edge weight is solved by the kernel affinity calculated with the PPK. The optimal normalized cut is recovered using a spectral approximation algorithm.



Step 4 - Clustering recovered

Figure 7.4: Clusters can of similar behavior are recognizable in this space using the probability product kernel.



Step 5 - Clusters merged and labels are assigned

Figure 7.5: Similar hosts are grouped together to form clusters for statistics-preserving crowd-based anonymization.

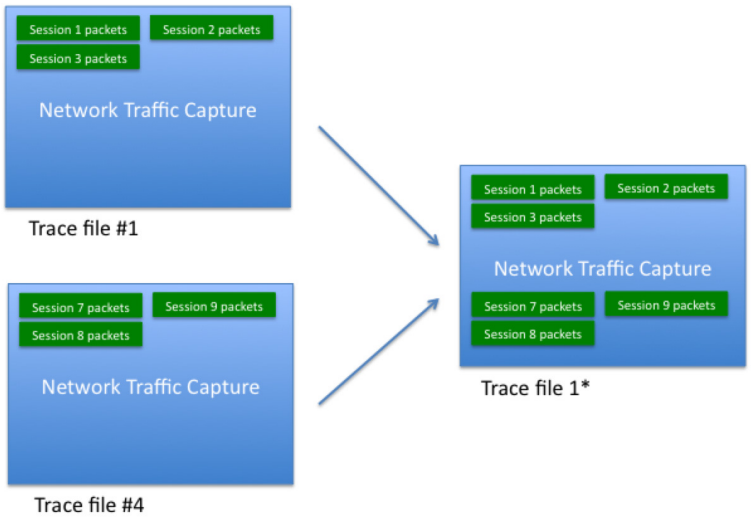


Figure 7.6: Anonymity by crowds is actualized through a packet/flow merging transformation on the appropriate hosts in the trace file.

7.1.1 Anonymity in the mixed traffic set

For realtime mixnets, the anonymization system induces end-to-end anonymity in that observers sitting on the edge of the circuit can only see traffic flowing into and out of the system but cannot match the source and destination pairs. Here, we analyze what anonymization by clustering achieves for the offline case.

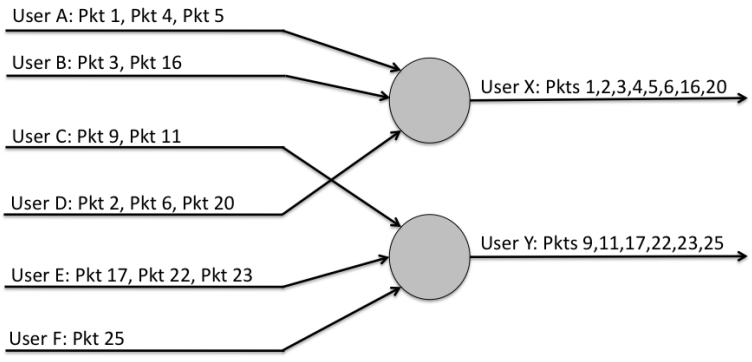


Figure 7.7: Mixing traffic amongst similarly behaving hosts.

Consider figure 7.7. Users A, B, and D are determined to behave similarly, as were users D, E, and F. Traffic for users A, B, and D are aggregated and reassigned a new identity

of user X. The same is done for user Y, and it now appears that two users generated the original traffic. Let unique sessions be identified by the labels Pkt1, Pkt2,...*etc.* We can assume that an observer has a segment of User A's traffic and therefore knows the exact characteristics of this session such as parameters of the TCP handshake, the initial window size negotiated, the duration, initial TCP sequence numbers, etc. In an unmixed packet trace the IP is simply obfuscated, the attacker now has the pseudo-identity for User A and can see all of the other sessions (Pkt4, Pkt5). Under our anonymization transformation, the attacker can only identify that User X is the pseudonym for User A. He cannot infer anything else about the traffic output from User X, such as how many members User X represents, and which sessions (represented by Pkt1,2,3,4,5,6,16,20) actually belongs to User A. Given that our behavior models track common services exercised as well as volumetric and time-series information, all of these sessions are quantifiably similar and difficult to separate. Further, he cannot infer whether or not all of the sessions that he observes are authentic and not synthesized, nor can he determine if they all originate from the source true source network and were not mixed across different network traces. Outbound information such as cardinality of the outbound peers is obfuscated as a property of merging the traffic, thus this signal is removed. Further, all IP addresses are obfuscated using a technique such as `CryptoPan`. Therefore, the anonymization has made it so that User X is functionally equivalent to a mix net exit node.

7.2 Stability of the algorithm

Given that our proposed algorithm is a stochastic-based method, the utility is partially conditioned on the stability of this algorithm. In order for PTM-based clustering of host behaviors to be a practical solution, the performance of the modeling, clustering, and synthesis steps must be predictable and consistent. We associate the study of these problems into the category of algorithm stability. An evaluation of such stability is presented in this section. A few important problems are discussed along with their corresponding proposed solutions. The first problem is how to determine the optimal cluster size k, and the second is the related problem of how to guarantee this minimum size in the clustering result. The

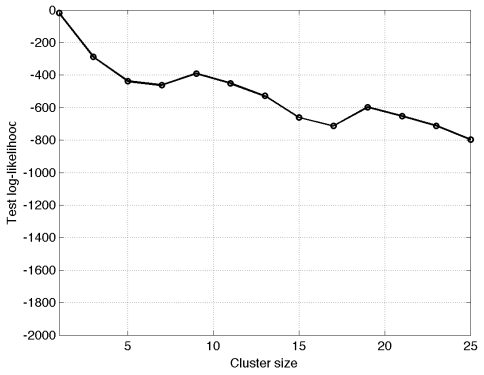
quality of the matchings, in terms of disparity between traffic-set sizes belonging to matched hosts, is examined, along with the predictability and stability of the synthesis algorithms.

7.2.1 Determining the optimal k

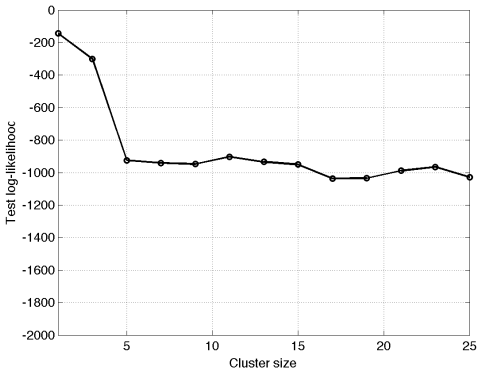
One of the foremost questions regarding this approach to anonymization is how to determine the optimal clustering size k. In practice – within this context and other similar anonymization approaches, such as micro-data anonymization – k is chosen by the user of the system in a way to minimize information disclosure about any individual measured entity. In micro-data anonymization, the mean and standard deviation may be used to determine – often in an ad hoc manner – the optimal cluster size.

Fundamentally, the motivation of establishing a cluster size is to obtain a favorable balance between the tradable in utility and anonymity. It is a reasonable assumption that the utility of the dataset is inversely proportional to the amount of modifications performed on it – an unmodified dataset retains all of the original utility properties, after all. As such, k should be chosen to be the minimum value that satisfies a measurable quantity of anonymity. What that measurement is, and how it can be used to obtain an estimate for k is studied in this section.

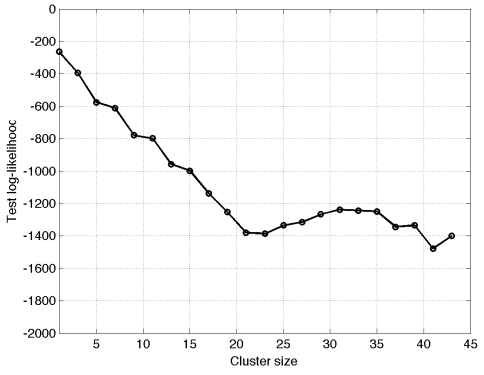
Our approach to estimating the tradeoff between utility and minimum cluster size is based on the likelihood estimate of the test data over a model trained on the mixed traffic dataset. Specifically, random segments of a host's traffic are extracted for use in the testing set. Then, that host's model is retrained using the merged data belong to that host as well as its nearest neighbors. The new model is used to evaluate the extracted test dataset from that particular host. A plot of the likelihood loss can then be used to estimate the amount of information lost or conversely, the anonymity gained. This simulates the effect of information loss by losing the exact labeling for that host. Essentially, it measures the loss of identity when merging into a crowd. It is the same scenario that an attacker who is trying to de-anonymize the dataset would observe – one where he knows the dataset is a mixture of multiple hosts. Using the likelihood value as opposed to classification accuracy is appropriate because classification is determined by the likelihood, therefore the the two values are strongly correlated.



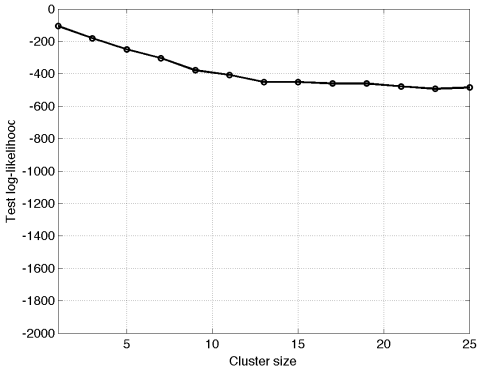
(a) Columbia Univ.



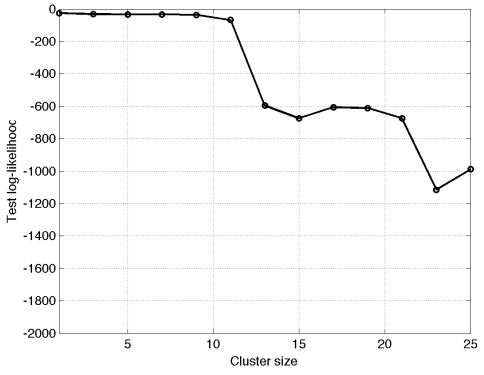
(b) George Mason Univ.



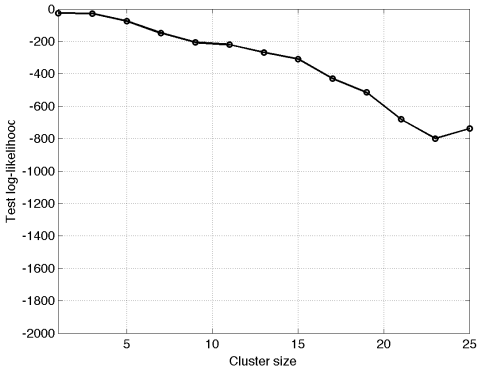
(c) Univ. Michigan



(d) Lawrence Berkeley National Labs



(e) West Point



(f) NSA

Figure 7.8: Measuring log-likelihood loss as a function of cluster size (k).

Figure (7.8) shows examples of this measurement. In this experiment, we obfuscate the host's traffic using PTM-measured nearest neighbors to generate the clusters of varying size. We measure the loss in the log-likelihood of the host's testing data when the model is trained with data that does not belong to the same class. The expected behavior is that the log-likelihood would continuously decrease, until the cluster has grown to the point where very non-similar hosts are added and the performance becomes chaotic. Before this point is reached, however, one might notice a fall-off point where adding similar hosts no longer affect the performance.

The data-fidelity loss is both stable and linear in the number of elements in a cluster; as more hosts are added to a cluster the fidelity of the data, as represented by the log-likelihood graphs, drops – as expected. This provides insights into necessary cluster-sizes: GMU, for example, achieve optimal clustering at size 5. The most evident example of this occurs in the GMU experiments where we see that the optimal cluster size is 5 – beyond this, the identity-loss (or conversely the anonymity gain) levels out. This allows us to measure anonymity gain and provides insights into how one can estimate the necessary size of the clusters.

7.2.2 Minimum-sized k guarantee when not using spectral partitioning

In the section that covered spectral clustering, we introduced the spectral partitioning algorithm, which guarantees a balanced clustering.

In the standard spectral clustering set up, a minimum cluster size is not guaranteed in the resulting labeling. In fact, singleton clusters can emerge; the frequency of which is based on the distribution of the data. Given the evaluation for cluster size discussed in the previous subsection, it is necessary to provide a guarantee for the minimum cluster size for the general case. This necessitates an additional processing step in the clustering procedure, in addition the spectral clustering. In this section we show how an agglomerative-clustering step can be utilized to provide this minimum k guarantee.

Agglomerative clustering is a hierarchical approach to clustering where pair-wise elements are recursively grouped from the bottom up. The algorithm proceeds in iterations where, within each iteration, pairs of datums are matched based on their similarity to each

other. The iterations are repeated until all datums are matched. In our problem setting, the datums are cluster labels outputted from spectral clustering. Thus, it is necessary to iteratively cluster pairs of clusters (not individual hosts.) A metric for cluster-similarity is then necessary.

One generative approach would be to retrain a single PTM model for each cluster using the traffic belonging to the hosts within that cluster $\theta_{\mathcal{A}} = \arg\max_{\theta} p(X \in \mathcal{A}|\theta)$. The PPK can then be used to match similarity between clusters:

$$S(\mathcal{A}, \mathcal{B}) = \mathcal{K}_{\text{PPK}}(\theta_{\mathcal{A}}, \theta_{\mathcal{B}}). \tag{7.1}$$

This can be done recursively in order to perform agglomerative clustering. This is an EM-like approach, however, it is computationally complex as the dataset would need to be processed on the order of $O(N \log N)$ times on average if the pairing tree is balanced, and $O(N^2)$ if it is unbalanced. However, re-training and re-computation of similarity scores is unnecessary. An efficient way to achieve the same hierarchical clustering is to take advantage of already existing kernel-based similarity comparisons found in the Gram matrix that was computed during the spectral clustering step. Recall that spectral clustering uses the eigenvalues of Laplacian matrix L, which is computed from the Gram matrix G. One can compare similarity between clusters directly by summing the pair-wise similarity scores between hosts within the two clusters, this is done by summing the rows and columns that correspond to the elements from the two separate clusters.

Let $\pi(i, j)$ return 1 if $x_i \in \mathcal{A}$ and $x_j \in \mathcal{B}$, and 0 otherwise.

$$\delta(i, j) = \begin{cases} 1 & : x_i \in \mathcal{A} \wedge x_j \in \mathcal{B} \\ 0 & \text{otherwise} \end{cases} \tag{7.2}$$

Finding the similarity between clusters of hosts corresponds to summing elements of the gram matrix $\mathbf{G}$:

$$\widehat{S}(\mathcal{A}, \mathcal{B}) = \sum_{i=1}^{N} \sum_{j=1}^{N} \mathbf{G}(i, j)\delta(i, j). \tag{7.3}$$

This method, in essence, compares the distance of all members of one cluster to all the members of another. An efficient way to do this is to extract the entries from the Gram

matrix where the rows correspond to the entries of one cluster and the columns correspond to the entries of another cluster. The sum of this reduced matrix gives the aggregate similarity between members of the two clusters. Performing the agglomerative clustering step is then straight forward, using iterative matching at each layer.

Figure (7.9) shows an example of this agglomerative clustering method on the Columbia dataset. Iterative merging of under-sized clusters is shown, with $k = 5$ set as the minimum cluster size. In the top left panel, the results of the spectral clustering procedure are shown; the figure shows a histogram of the number of elements within each cluster. (For the scope of this experiment, no effort was made to adjust the algorithm to yield more favorable balanced clusters; in practice, proper regularization of the probability-product kernel can be used to produce more balanced cluster sizes naturally.) The subsequent panels show the steps of the agglomerative clustering procedure as the clusters are merged with their nearest neighbors and their host counts are aggregated, until no cluster is left with less than 5 members. The Columbia dataset was chosen to demonstrate the stability of this algorithm since the true-labels for the hosts are known. Therefore, we can examine the results of the final clustering using the host names.

Figure (7.10) shows the results of spectral clustering with an agglomerative matching fix-up phase. In this resulting dataset, we can see all of the web servers are clustered together in cluster 4. This cluster was missing "web3157" in earlier evaluations, but this host was matched to this cluster in this hierarchical matching scheme.

In terms of runtime costs, given that the components of the Gram matrix are used, the algorithm runs in time proportional to the number of clusters (which is very small compared to the number of hosts). Since no retraining or re-evaluation of the dataset is necessary, and only comparisons of values within an $N \times N$ matrix is needed, this procedure as negligible overhead to the original clustering function.

7.2.3 Parity in the clustering

Thus far, examinations of clustering stability has focused no the host-level entity, comparisons were based on host label matchings and not the amount of traffic each host possessed. Stability in behavior matching, however, is conditioned on the host as well as the volume of

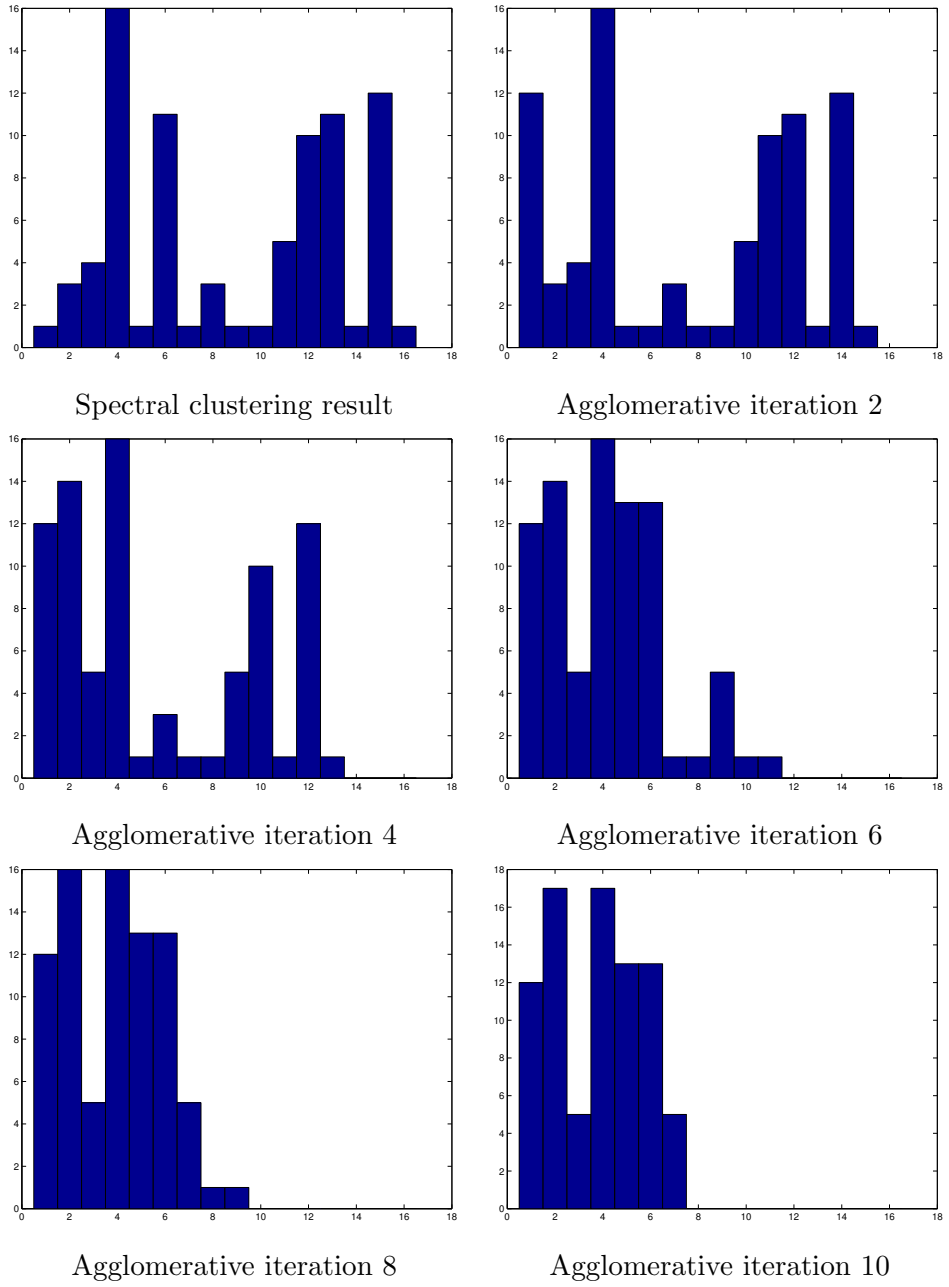


Figure 7.9: Agglomerative clustering to guarantee minimum cluster size ($k = 5$.) Distributions of cluster sizes at each step is shown.

(1) leaf
(1) metro23
(1) cityhall
(1) forest
(1) multitouch
(1) unknown
(1) unknown
(1) mail

(2) dhcptemp33
(2) dhcptemp48
(2) pachira
(2) dhcp11
(2) spectral
(2) fdr
(2) metro20
(2) goya
(2) uribe
(2) irtdesk1
(2) irtdesk5
(2) coulomb03
(2) ng911-db2
(2) picard
(2) hamming
(2) green
(2) cutunes
(2) klimt
(2) apadana

(3) walker.win
(3) carter.win
(3) dhcp72
(3) metrorail
(3) workstudylaptop
(3) water
(3) kkh
(3) fuji
(3) dhcp85
(3) furst-pc
(3) sekhmet
(3) zapp.win
(3) fry.win

(4) web1
(4) web2
(4) web3
(4) web4
(4) web3157

(5) game
(5) gabriel
(5) raphson
(5) dhcp25
(5) guajira
(5) ense-client
(5) sos1
(5) sos2
(5) sos5
(5) sos6
(5) sos9
(5) ense1
(5) ense2
(5) hikari
(5) lagrange
(5) bailey
(5) elba

(6) cs
(6) animal
(6) raphael
(6) templar
(6) baker.win
(6) dhcp64
(6) neale.ccls
(6) cluster00.ncl
(6) pyromania

(7) honda
(7) dhcptemp27
(7) manata
(7) corsica
(7) dhcp30
(7) dhcp49
(7) racecar
(7) kathy-pc.win
(7) tardieu-desktop
(7) boyacipc2

Figure 7.10: Results of spectral clustering with agglomerative merging of the resulting clusters. The minimum cluster size was set to 5.

traffic belonging to that host. For example, if a host containing 1 million packets is matched with a host containing 100 packets, the disparity in data volume is naturally going to have an adverse effect on the anonymization, regardless of how similar the behaviors of these two hosts may be. Therefore, this section examines the parity in the traffic volume between the host mixtures. While traffic volume is not explicitly modeled in the PTM, some estimate of volume is intrinsically captured in the emissions model which measure the average size of the flow connections, though the total number of connections are not tracked. As the evaluations show, similarity in host behavior is correlated with traffic volume.

If a similar-behavior based approach is used to de-anonymize the merged trace, then, without true labels on the traffic segments, the attacker draws portions of the mixed traffic for training. Anonymity is proportional to the probability of drawing any given host's packets from the mixed source. Let this be defined naively as the ratio of packets between those belonging to a particular host and those of the entire cluster. Let c_i represent the count of the number of packets for host i in a cluster. The probability associated with each host is then associated with the ratio, and the measure of parity is based on the Shannon entropy in the packet-count distributions within a cluster:

$$p_i = \frac{c_i}{\sum_{j \in \mathcal{A}} c_j} \tag{7.4}$$

$$H(\mathcal{A}) = -\sum_{i=1}^{|\mathcal{A}|} p_i \log p_i. \tag{7.5}$$

And the expected value of H for the k clusters is then used for the parity metric of the clustering algorithm.

$$H^*(\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_k) = \frac{1}{k} \sum_{i=1}^{k} H(\mathcal{C}_i). \tag{7.6}$$

Figure (7.11) shows the stability of the matching with respect to the packet count parity. Average packet-count disparity vs. number-of-clusters: the ability of the system to match up even-sized traces with each other as the number of clusters increases. In (a) we measure the homogeneity of packet-counts of members element. This means that if a host with a large set of associated data is mixed with one that has few data, the score would be low.

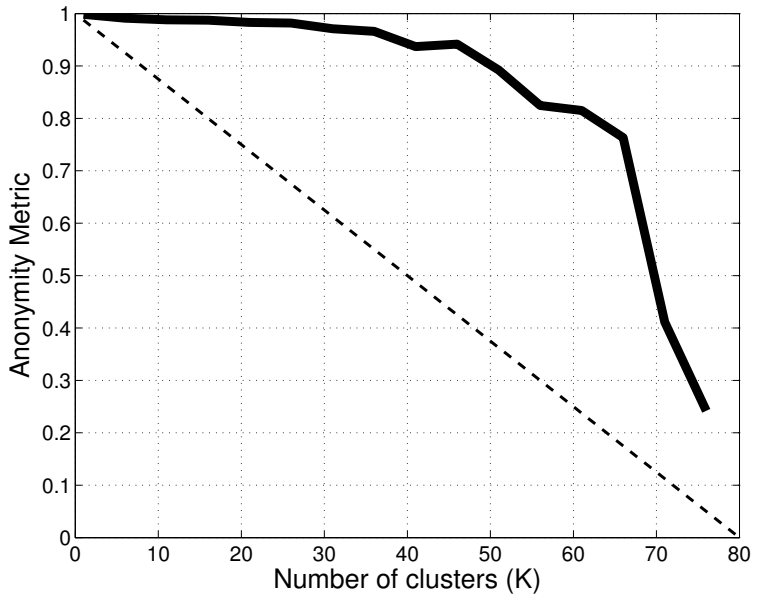


Figure 7.11: Average packet-count disparity vs. number of clusters in Columbia dataset.

A baseline of random mixing is shown in the dashed line. As the figure shows, our method maintains good cluster parity until the cluster sizes approach the size of the entire dataset and bad mixes are forced.

7.2.4 Traffic interpolation and synthesis stability

Among the anonymization transformations is the ability to synthesize new data. The use of statistical sampling and replacement of true values with these samples is used in micro-data anonymization, and has been studied in the case of differential privacy. In this section, we evaluate the stability performance synthesizing new traffic using the behavior-based models.

We use a *visual* similarity-measurement technique to quantify the fidelity of the synthesized data. This is done by plotting points in a 2-D plane in a way such that each point corresponds to a synthesized data sequence, and the distances between points in this 2-D plane is proportional to the distances between the corresponding traffic samples in the original space. This technique is known as "embedding," and we have developed methods for network-data embedding in our prior work [**?**]. In this experiment, we began with two classes of behavior: one modeled after a web server (denoted by "web") and the other mod-

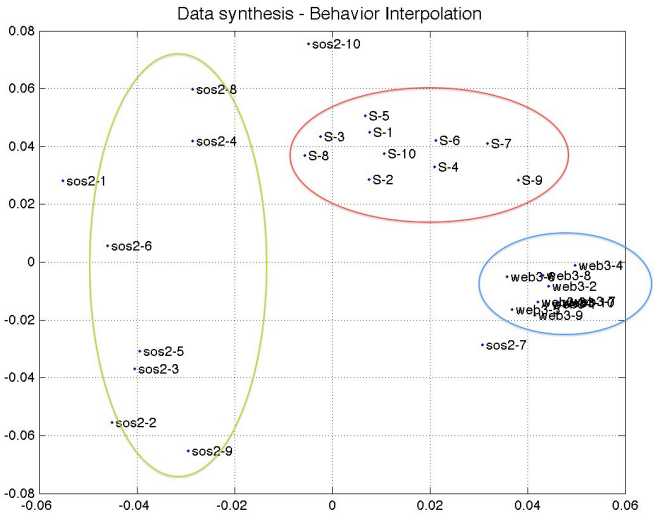


Figure 7.12: Behavior interpolation: Two clusters of behavior profiles are embedded ("sos" and "web") along with a new class of behavior ("S") that interpolates between these two models.

eled after a compute server (denoted by "sos"). A Parzen-based switchng model, similar to the one described in equation (5) is used to synthesize samples of 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. We use the kernel principle component analysis with the PPK-PTM kernel to achieve this embedding.

7.3 Self-similarity and behavior shaping

The concept of traffic shaping has been studied in the context of anonymization for many years. The fundamental goal behind traffic shaping is to obfuscate any statistical or content characteristics from the traffic that might leak information about the identity of the anonymized entity, such as a host or a communications stream. Traffic shaping has been discussed in the context of onion routing networks since 2001[Syverson *et al.*, 2001] where data padding and other transformations are investigated to provide cover traffic for anonymized communications channels. Recently, traffic morphing for offline data was discussed by Wright *et al.* in [Wright *et al.*, 2009].

Traffic shaping is mostly used in tandem with statistical models where a measurement of behavior or other characteristic is quantifiable. Given the ability to model and measure such behavior, it is a natural extension to then use this measurement to reshape those statistical properties to achieve some specific goal: either blending into desired goal or add enough noise such that sufficient divergence from a target model is achieved. In the packet trace setting, this is typically done by modifying the original packet streams with transformations such as adding, removing, modifying packets.

An example of where such methods are applicable would be identifying users that have very distinctive behavior patterns, such as a person who generates a very high amount of FTP traffic. This individual might be easily identifiable even if their IP address was obfuscated using CryptoPan or a comparable method. However, changing the IP address of the host does not mask that host's behavioral profile and, using the same behavior models that were presented in this thesis, that user is detectable in the IP-anonymized result. Some transformation which would improve anonymity and remove this overt signature would be to systematically drop FTP packets from that host's traffic stream, or insert synthesized FTP traffic to similar hosts, such that the signal-to-noise ratio is diluted for that particular host.

7.3.1 Self-similarity measurement

This section and the following discuss traffic shaping in our anonymization context. In order to reshape behavior, there must be a method of measurement to act as the baseline for evaluation – a way to measure the changes induced by any obfuscation transformation. For anonymization in web-traffic, the distribution of packet sizes were used [Wright *et al.*, 2009]. For our research purpose, we are interested in an overall behavior profile for the host. For a baseline measurement, we use a measure that we refer to as the "self-similarity" measure. This is a measurement of the likelihood of a host's traffic as evaluated by that host's own behavior model.

Self-similarity is determined by moving a siding window of a set size across a host's traffic dataset, and at each step evaluate the likelihood of the traffic within that window using that host's model. Let $\mathbf{x}[i : i + w - 1]$ represent a window of traffic of length w for

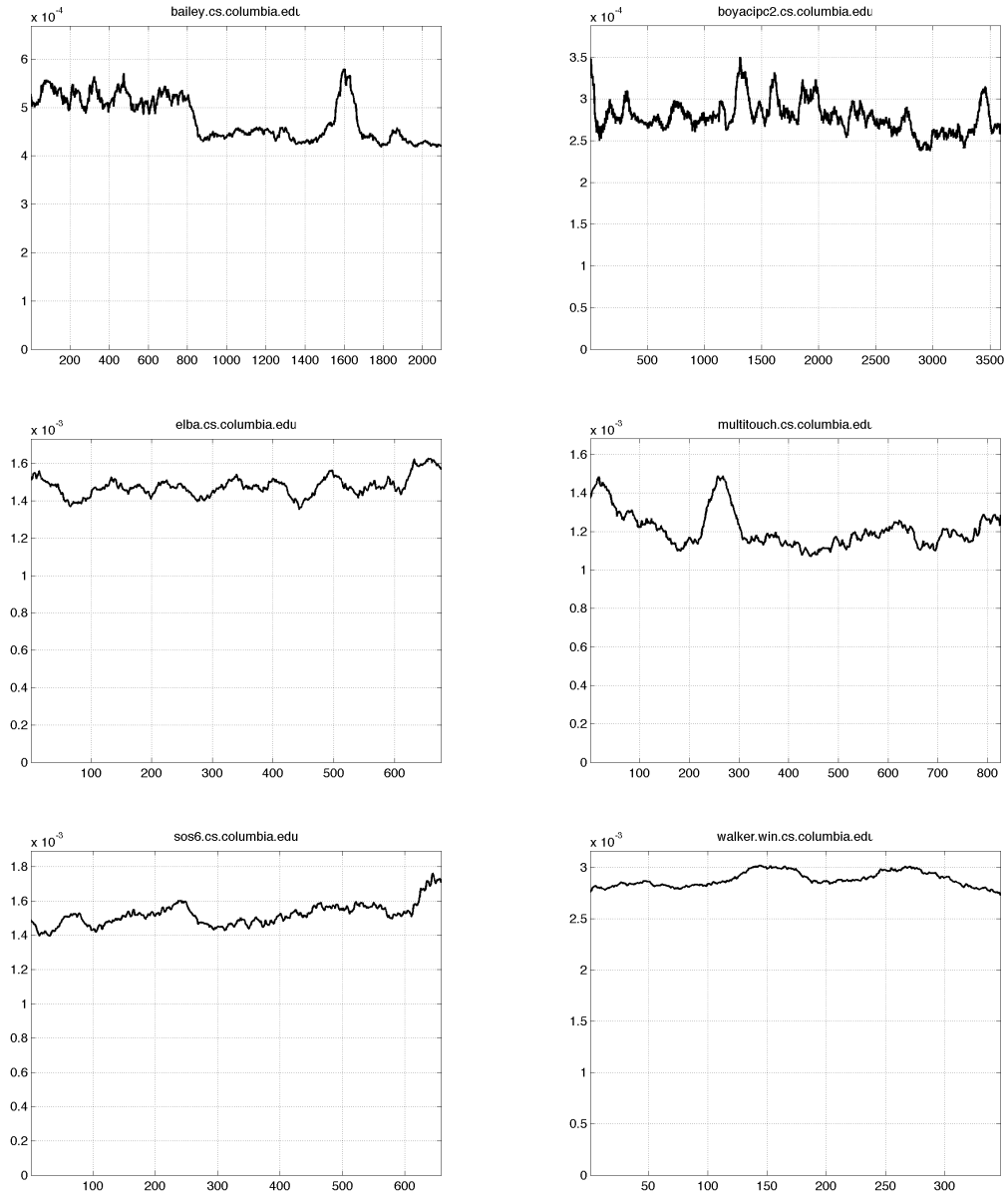


Figure 7.13: Self-anomaly of different hosts across time. Calculated as normalized negative log-likelihood score of data within a sliding window swept across the traffic.

traffic sample $\mathbf{x}$.

$$\psi_i = p(\mathbf{x}[i : i + w - 1]|\theta) \qquad \forall i = 1, \ldots, N - w. \tag{7.7}$$

Generally, we are interested mostly in evaluating anomalous behavior. Such traffic present statistical signatures that could be a source of privacy leak. The measure is inverted therefore, to use the negative of the loglikelihood estimate, in order to reflect a measurement of self-anomaly. Figure (7.13) shows this plot of self anomalous behavior for various hosts within the Columbia dataset. This measurement is used in the subsequent subsections to measure the effects of proposed traffic-shaping transformation techniques.

7.3.2 Traffic smoothing: anomaly removal

A host's behavior profile may not always appear consistent, as some of the self-similarity plots show. It is normal for hosts to exhibit periods of abnormal behavior with respect to its own model. This happens sometimes due to outside influence. For example, if a machine is configured to respond to failed connection attempts then a port scan would elicit a response that would appear anomalous to normal behavior patterns. Such a scan can be used as an active fingerprint injection method to embed a noticeable signature into the dataset that can be retrieved later to identify the host within the anonymized dataset. For this and similar reasons, it is desirable to have a means to remove measurable anomalies from the traffic dataset.

Figure (7.14) shows a traffic smoothing procedure that removes anomalous traffic that is one standard deviation above the mean from the host's traffic dataset. The result of these removals is observable in the corresponding self-similarity plots for various hosts. The granularity of the smoothing is a function of the window size w described in equation (7.7).

7.3.3 Traffic perturbation: anomaly injection

Removing anomalous traffic is only one-half of the traffic-shaping procedure. The second part is the insert of traffic with targeted similarity scores. This allows one to fully manipulate the behavior profile of the host in any desirable way. One way of inserting such traffic is to synthesize this traffic using the techniques discussed in section § 6. Since the state models

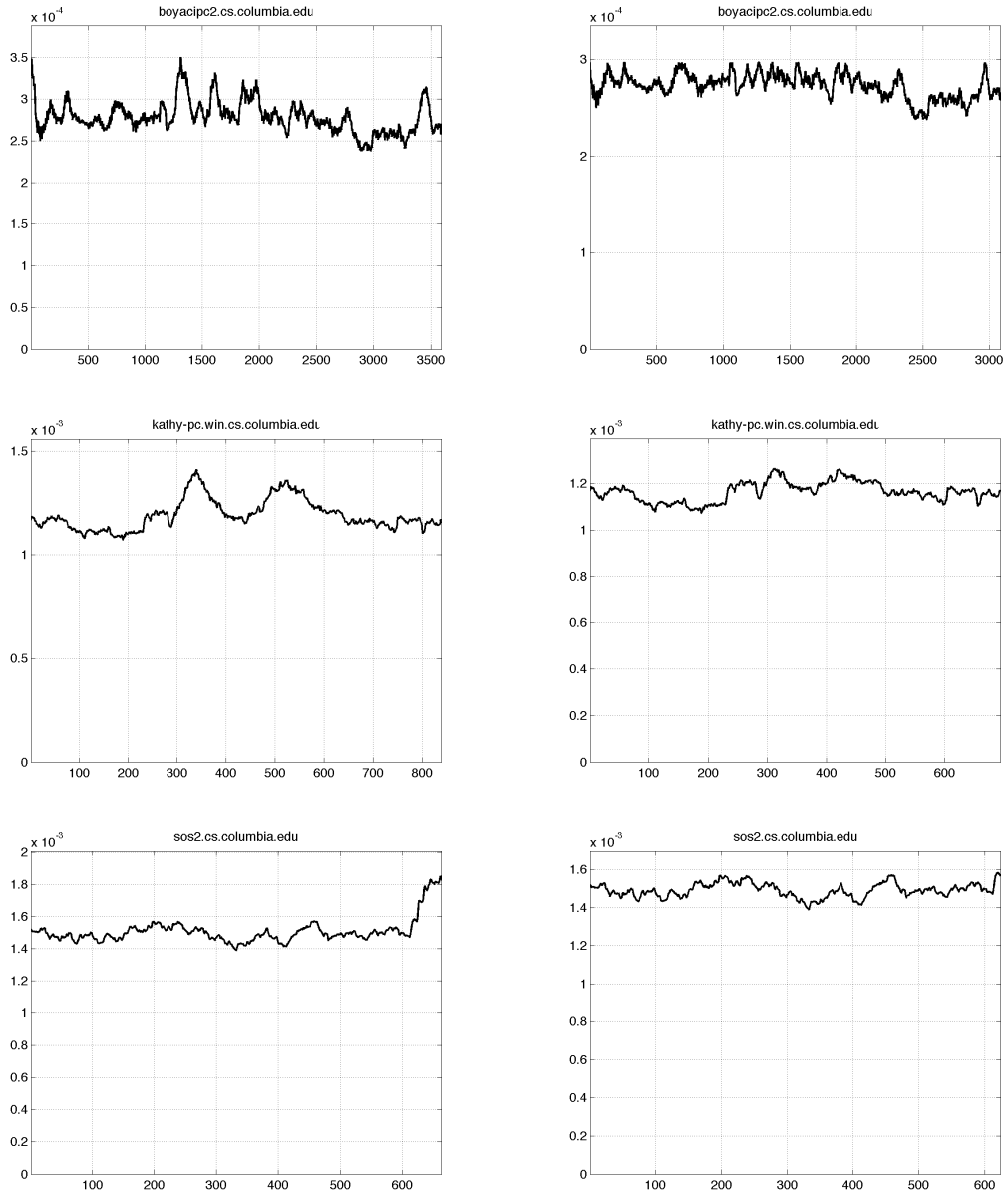


Figure 7.14: Traffic smoothing: self-anomalies – segments of traffic with anomaly scores above 1 standard-deviation from the mean are removed. Original is shown on the left, smoothed traffic is shown on the right.

of the PTM are directly tied to the port values, adding white noise to the models is a simple way of generating similar behavior models. This can include perturbation of the transition matrix and the emissions models which measure the volumetric distributions.

Using synthetic data may not always the most attractive solution, however. In many cases, using authentic data is more desirable. Re-shaping the traffic profile of a host by inserting other pieces of authentic data may initially seem difficult, though a simple method is available. This method is to rank individual traffic segments of similar hosts using a sliding window, sorting segments based on their similarity score and using this as an index into determining which portions of traffic to insert the host's traffic that is being reshaped. Let $\widehat{\theta}$ represent the trained PTM model of the host who's traffic we are reshaping, and $\mathcal{H}$ represent traffic samples from a set of hosts.

$$s_{i,j} = p(\mathbf{x}_j[i : i + w - 1]|\widehat{\theta}) \qquad \forall \mathbf{x}_j \in \mathcal{H}, i = 1, \ldots, |\mathcal{H}_j| - w. \tag{7.8}$$

The above equation essentially calculates a matrix of similarities of segments to traffic for every host in the dataset. In practice, this exhaustive computation is not necessary – only the most similar hosts would need to be evaluated. Note that $\mathcal{H}$ does not necessarily have to be the same set of hosts from where the target is chosen from – it is possible to cross-insert traffic from multiple datasets among each other.

Figure (7.15) shows an example of inserting various ranges of anomalous traffic into the traffic of a target host. Only a single anomalous segment is inserted in each example in order for the changes to be easily observable in the plots. As the figures (7.14) and (7.15) show, reshaping the behavior profile of a host is straight forward using the PTM models.

7.3.4 Nearest-neighbor matching for traffic smoothing

Further still, by partially mixing traffic among different hosts, we can also achieve an interpolation effect on the profiles. This allows us to essentially make users behavior 30% similar to one specific profile and 70% similar to another, or some other desired ratio. This is done by taking portions of traffic from the desired profiles and mixing it with the traffic of the target host. Replication of existing sessions with added perturbations can be used

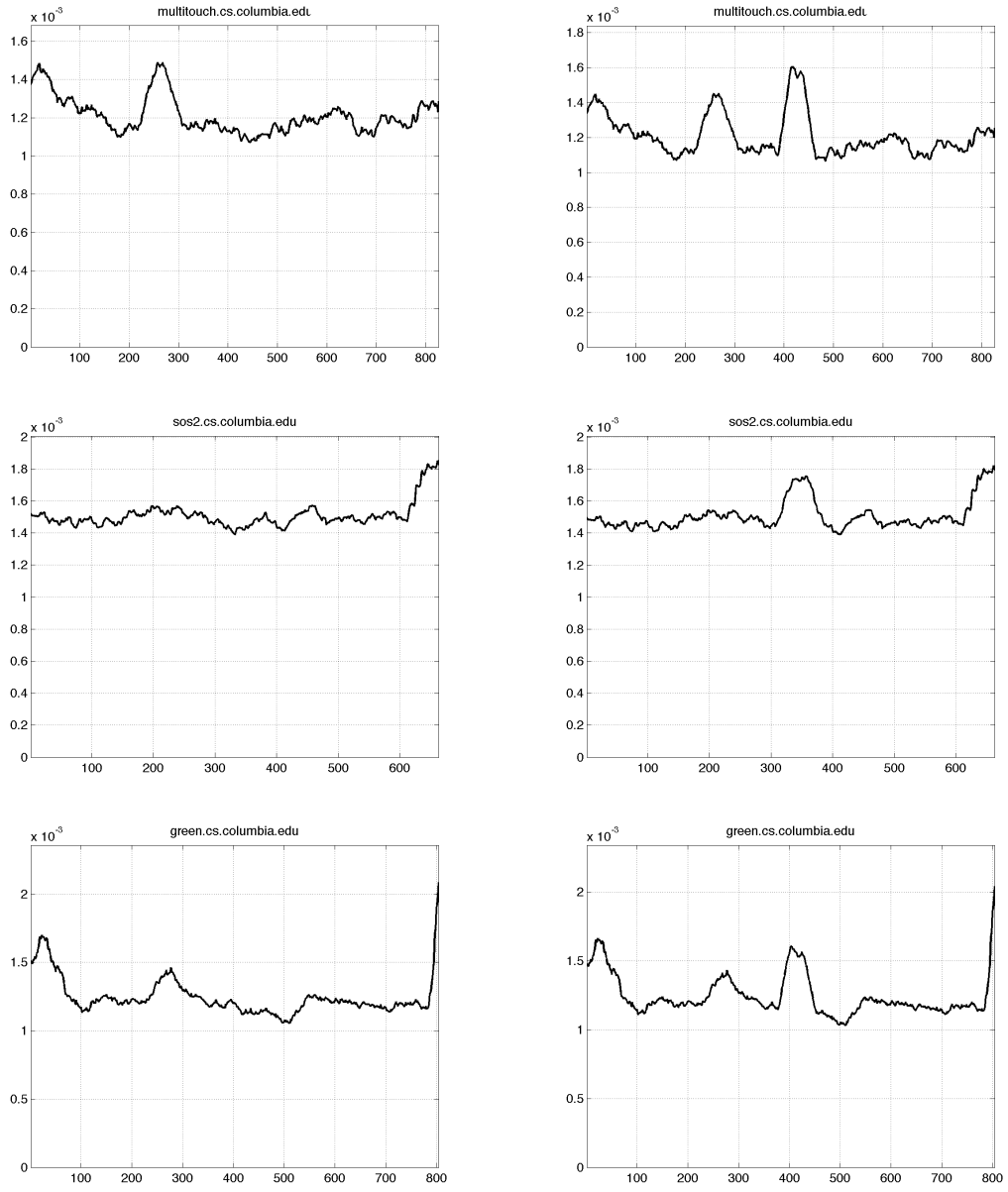


Figure 7.15: Traffic shaping: inserting anomalies – segments of traffic are modified to induce self-anomalies. Anomalies are inserted at the half-way mark of each traffic set.

to dynamically shift the behavioral patterns of groups of hosts or individuals, to further enhance anonymity, reduce outliers, smooth the dataset, and more, all the while allowing aggregation statistical information to be preserved and information-loss minimized.

7.4 Technical details of implementation

The techniques of merging and reshaping traffic has been described at a high level in this thesis. This section provides a description of the technical details of implementation involved in packet-trace manipulation. For statistical datasets such as Netflow records, merging traffic is trivial as only rows of entries need to be exchanged and modified. For packet data, the modifications involved are more complex as all changes to the packets must be undertaken without destroying the TCP sessions that they represent. In addition, some related technical challenges with merging, as well as post-processing steps in anonymization are discussed.

7.4.1 Technical challenges in merging traffic

Overlapping TCP windows: TCP sessions between two hosts on the same pair of source and destination ports cannot overlap in time, as this would mangle the individual connections and render both packet streams incapable of being sessionized. On occasion, this event can happen when merging traffic sessions, especially since overlapping sets of ports used between different hosts indicate a higher similarity measurement. When this happens there are two options available to fix this problem: first, the time-window of one of the sessions can be increased so that they no longer overlap. Second, one of the ports on either session may be modified to break the overlap; usually it's best to identify the session using an ephemeral port and modify that to induce minimal change in the data. Modifying the port in any other case can be done by incrementing its value.

IP and link-layer address obfuscation: The IP and Ethernet headers for the hosts must be modified to be consistent with the session-layer modifications to the TCP packet headers. This includes ensuring that the obfuscated IP addresses derived post-merging, are

consistent in the different protocol layers, as well as the MAC address.

IP ID linearization: The IP ID field is used in TCP/IP for packet fragmentation related issues. When a packet is fragment during routing it needs to be reassembled at the end point. In order to track the fragmentation, a unique 32-bit ID is assigned to each IP packet. This ID is linearly increasing and independent for each host. This property is a TCP/IP specification and is independent of host behavior. If left untouched, the incrementing sets of IP ID values may reveal the total number of hosts within a merged cluster. Consider the following sequence of observed IP ID values: 1,2,17,18,19,154,4,20,155,156,157,5,21,158. Sorted, this values become: [1,2,3,4,5],[17,18,19,20,21],[154,155,156,157,158] and it is then obvious that traffic from three separate host were combined. This technique was first described by Bellovin as a means of counting hosts behind a NAT [Bellovin, 2002]. In order to prevent this form of analysis, the IP ID fields of the packets must be linearized. Thus, 1,2,17,18,19,154,4,20,155,156,157,5,21,158 should become 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15. The base-value of "1" should also be obfuscated so that if an attacker has the other side of the connection, matching hosts by IP ID sets is not possible.

Reinitializing TCP sequence numbers: TCP sequence numbers are used to keep track of the moving window in the TCP connection. These values are randomly initiated and incremented during the session. In order to prevent traffic-matching, all TCP sequence numbers for each session should be shifted by a random offset.

Post merging self-connections: It is sometimes possible that two hosts who have sent traffic to each other may be selected to be merged. This is problematic as it would cause self-to-self traffic instances to show up in the merged dataset. The simplest way to fix this issue is to scan for self-to-self connections in the merged dataset and drop such connections entirely.

Post-merging transformations: The behavior-based anonymization transformation described in this thesis is meant to be used in tandem with other anonymization frameworks

such as `tcpmkpub` or `PktAnon`. The prefix-preserving quality of the `CryptoPan` algorithm makes IP-address selection for the merged clusters simple. One can simply chose the IP address of the first member of the cluster. This, along with a prefix-preserving transformation allows preservation of the subnet structure of the network. Given that packet header values are modified in this anonymization procedure, all checksums must be recomputed. Functions for these transformations are available in the aforementioned anonymization frameworks and can be used after behavior-based anonymization.

7.4.2 Runtime

Training algorithm: The PTM training algorithm consists of two parts. The first is the transitional probability model for the various states. Given that the transition probability is updated with each flow sample, the total runtime growth of this component is $O(N)$ where N is the length of the data, or rather the number of flow entries. The second component of training consists of the EM-based estimation of the emissions models for each state. This algorithm runs in (SkN) where S is the number of states and k is the number of iterations it takes for EM to achieve convergence. For scalar data, which is what this model operates on, k is usually very small and as Table (7.3) shows, S is also typically very small. Therefore we have $(k, S << N)$, and the runtime growth of the entire training algorithm grows close to $O(N)$ time. In practice, training full PTM models for over 82 hosts, representing the majority of the traffic observed in the dataset, with over 1.6 million instances of extracted flow statistics, took **0.7 seconds** in an unoptimized Matlab implementation, on a 3Ghz machine.

Likelihood Evaluation algorithm runs in $O(N)$: Likelihood evaluation consists of iteration through each element of the flow entries once. Therefore the algorithm runs in $O(N)$.

Probability product kernel $O(S_1 S_2 T)$: The runtime of the probability product kernel depends on the number of states for each model S_1 and S_2, which, as previously shown, is very small. T is a parameter control operator which is generally fixed at a small value. In our experiments it was fixed at 10. Given that most hosts contain a very small number of states, the growth rate of the PPK is negligible and can be considered fairly constant.

Bottlenecks: The main bottleneck in the proposed anonymization framework comes from the cost of processing the packet data. Our algorithm requires flow-level statistics about the hosts traffic. Therefore the processing time mostly is conditioned on the performance of tools such as `tcptrace`[Ostermann, 2003], `softflowd` and `nfcapd`, and other similar tools which can be used to extract flow-layer statistics from PCAP traffic. Within the machine learning components, the bottleneck resides in the eigenvalue decomposition algorithm, though this cost is also negligible (runtimes in seconds) compared to the PCAP processing costs.

7.5 Preservation of statistical properties

This section discusses the influence of anonymization on the statistical features of the data. The various components of the anonymization transformation are examined, and their effect on the dataset is discussed. We make the distinction between local and global statistics as those pertaining to the individual hosts and those pertaining to the entire dataset. Specifically, we consider the effects of merging and synthesis on these statistics, as these are the obfuscation techniques that induce the most amount change into the data. The following sub-sections discuss these two aspects in detail. Specifically, we consider the effects of this transformation on independent statistics, anomaly detection, behavior statistics, content, as well as communications structure of the network.

7.5.1 Independent statistics and content

The specific effects of merging and synthesizing data are discussed below. These are the common characteristics that are typically evaluated during network research, such as aggregate rates, volumes distributions, and content feature such as path MTU and intrusion detection. The preservation of these properties is important in order to maintain the utility of the dataset. Different transformations, such as synthesis or traffic perturbation, will induce distinct changes on the data, and how these functions are used – like other similar anonymization systems [Gamer *et al.*, 2008; **?**] – will depend on both the content of the

network traffic and the policy of the source provider. This section discusses some invariant properties in the anonymization system.

Volume information: Global volumetric information is completely preserved in the merging function. Given that merging merely entails re-assigning labels (*i.e.* IP addresses) to the individual hosts, the header and payload contents and aggregate statistical information such as packet counts within the traffic are not changed. Therefore merging has no effect on volumetric information. With regard to synthesis, the sizes of the synthetic flows are determined by the emission-models for each state and the transition-distribution for these states properties that are trained over actual traffic. Given the fact that both distributions are normalized, the linearity of expectations implies that the overall statistical-mean of the flow-size distribution will remain unchanged.

Port values: This feature refers to the range and distribution of port values observed in the dataset. This reflects on the distinct types of services used by the hosts within a network. Merging, as in the previous case, has no effect on the port-value distribution. When using synthesis, the distribution is entirely consistent with the training traffic, in that no unobserved ports are inserted into the dataset, and the values that are inserted are done according to the same distribution as the original model. The transition probability model of the synthesis algorithm is normalized – this means that the distribution of states (ports) is maintained in the synthesis.

Session characteristics: Network-implementation-oriented characteristics such as the distributions of path MTUs, and the corresponding maximum window sizes, are not effected by the merging transformation, as these are network content characteristics which are driven by the properties of the network and device implementation, and are not based on host behavior. From the perspective of the analyst, the set of network sessions that were observed to originate from several different hosts now appear to originate from the same host.

Number of hosts: By merging hosts, the overall number of hosts within the dataset is naturally reduced. In general, the number of hosts in the anonymized dataset will be close to N/k, where k is the cluster size. If the balanced spectral partition method is used, then this value is guaranteed. The same effect applies to evaluations of the number of distinct devices on the network. Given that header information must be modified to be consistent with the merging steps, the distribution of network devices with respect to unique content signatures will be reduced in the same ratio.

Payload content: Packet payload content is not handled by the proposed anonymization framework. This is because most network data is typically not released with payload content, and if such information is available the standard obfuscation technique is to hash this content or to remove it from the dataset entirely.

7.5.2 Preservation of behavior characteristics

This subsection focuses on the specific higher-level behavior characteristics of the network that are preserved. Two behavior-oriented host measurements are presented: the heavy-hitters distribution problem and a more general anomaly-detection problem. In both instances, we show the overall shape of the distributions are preserved before and after anonymization.

Heavy hitters distribution: The protocol-transition model uses volumetric information in the emissions models for each state. Though the volumetric information that the emissions model track relate to the flow-level statistics of the traffic, there is a positive correlation between the average flow-size and overall volume of traffic generated by a host. As such, volumetric information is abstracted in the PTM-tracked behavior profile for a host, and can be preserved when similar hosts are merged together. Figure (7.16) shows the volumetric distribution for the hosts within the Columbia University dataset. The figure shows volume distribution of heavy hitters on the network before and after the merging transformation. The results show that the overall shape of the distribution is maintained under behavior-based clustering. By comparison, the dashed line indicates the result of random

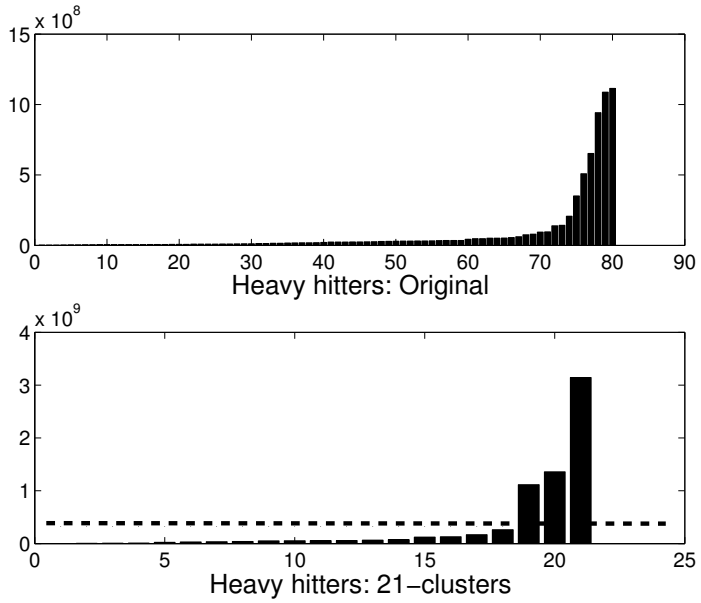


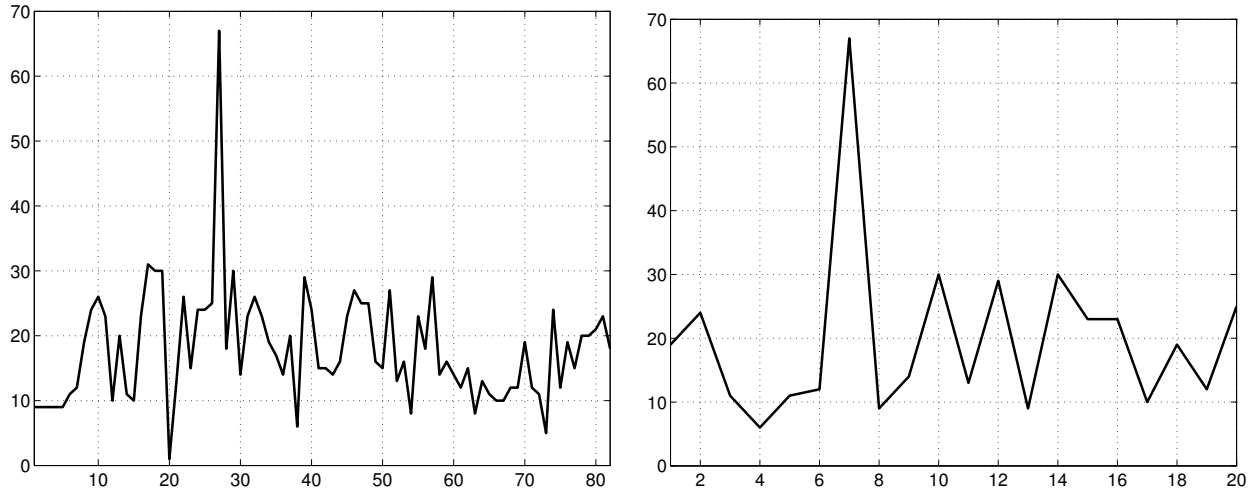
Figure 7.16: Preservation of the traffic-volume distribution.

mixing.

Services/port distribution: The port-value distribution tracks the distinct number of ports observed in the traffic for each host. This measurement summarizes the diversity in the types of services and protocols utilized in the dataset. Figure (7.17) shows the port distribution for the hosts within the Columbia University dataset before and after anonymization. As the figures show, the overall shape of the distribution is preserved.

Anomaly detection: At a host-behavior level, we would like to be sure that anonymization the dataset does not remove statistical anomalies which other behavior-based measurement systems might be interested in. Anomaly detection is used frequently in behavior-based IDS systems [Wang *et al.*, 2005; 2006; Song *et al.*, 2009], thus preservation of behavior anomalies is an valuable feature in the system.

As a general AD system, we use the kernel-density estimation (KDE) procedure described in § 5.6.2 to build a non-parametric model for our test network. Without making



(a) Before anonymization (b) After anonymization

Figure 7.17: Preservation of the shape of the unique-ports distribution. (a) 82 hosts are shown in the original traffic, (b) 20 pseudo-hosts are shown in the merged traffic.

parametric assumptions on the host behavior distributions, KDE is the most effective agnostic non-parametric density estimation approach, and extends naturally from our kernel derivations. Let $\Theta = \{\theta_1, \theta_2, ..., \theta_N\}$ represent the set of estimation PTM parameters, the KDE estimate is then:

$$F(\theta*, \Theta) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{K}(\theta*, \theta_i) \tag{7.9}$$

$$F(\theta) = \frac{\sum_{i=}^{N} \mathcal{K}(\theta, \Theta_i)}{\sum_{\widehat{\theta} \in \mathcal{D}} \sum_{i=1}^{N} \mathcal{K}(\widehat{\theta}, \Theta_i)}. \tag{7.10}$$

Where $\mathcal{K}$ represents the probability product kernel. As the equations show, only the pair-wise kernel estimations are necessary. Using, the PPK, we can bypass the need to evaluate similarity scores over the entire dataset and solve the estimate using the model parameters, making this a very fast algorithm. Since $F(\theta)$ returns a normalized KDE likelihood score for a particular host, we invert this measure to obtain an anomaly score: $S = 1 - F(\theta)$. To test anomaly detection we detect all

In this experiment we trained a PTM on each host within the Columbia University dataset and clustered them into 15 different clusters. As Table (7.1) shows, an anomaly was

$(s > \mu + \sigma)$ Anomalous host	$(s > \mu + \sigma)$ Anomalous cluster
dhcptemp33.cs.columbia.edu	fdr.cs.columbia.edu
	dhcptemp33.cs.columbia.edu
	picard.cs.columbia.edu
	klimt.cs.columbia.edu

Table 7.1: Anomaly detection in original and merged dataset.

detected in the original traffic dataset by finding hosts with anomaly scores one standard deviation above average for S. On the right, the anomaly detection procedure is repeated after the traffic belonging to the elements of the individual clusters has been merged. The anomalous cluster is returned this time, and the members of that cluster are printed using the original labels. As this example shows, the anomalous host is found once again. Thus, anomaly detection is preserved in this example. Note that in this example, the original labels are available. In practice, it wouldn't be possible to know how many hosts was in that cluster, or how to segment the traffic so such that he original anomalous host is identifiable.

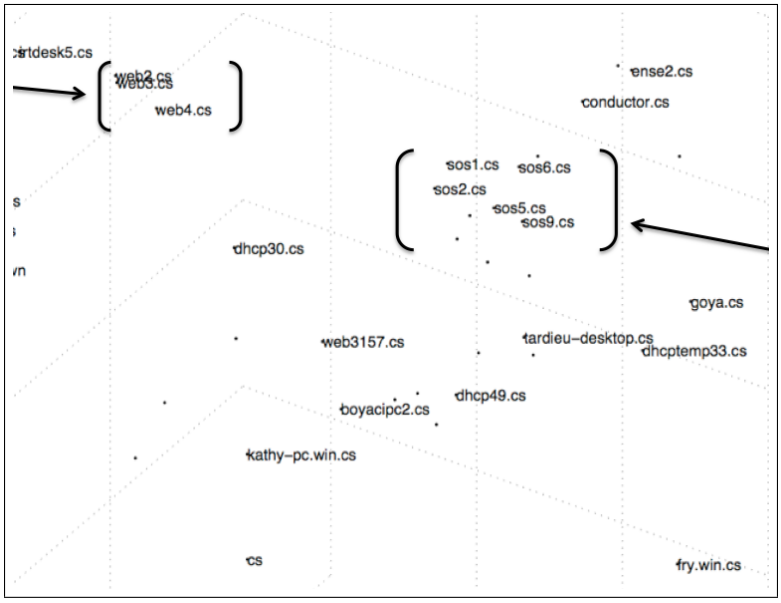
Of course, anomaly detection depends entirely on the dataset. In order to demonstrate generalizability, more than one example is necessary. Therefore, instead of using the score of one stand deviation above the mean, we simply rank the anomalies and return the top elements.

Table (7.2) shows the result when retrieving the second most anomalous behavior from the distribution. Note here that the second host is not considered anomalous by the standard measurement, nevertheless, when retrieving the labels of the second most anomalous cluster in the clustered dataset we see this host appear in the result, as expected.

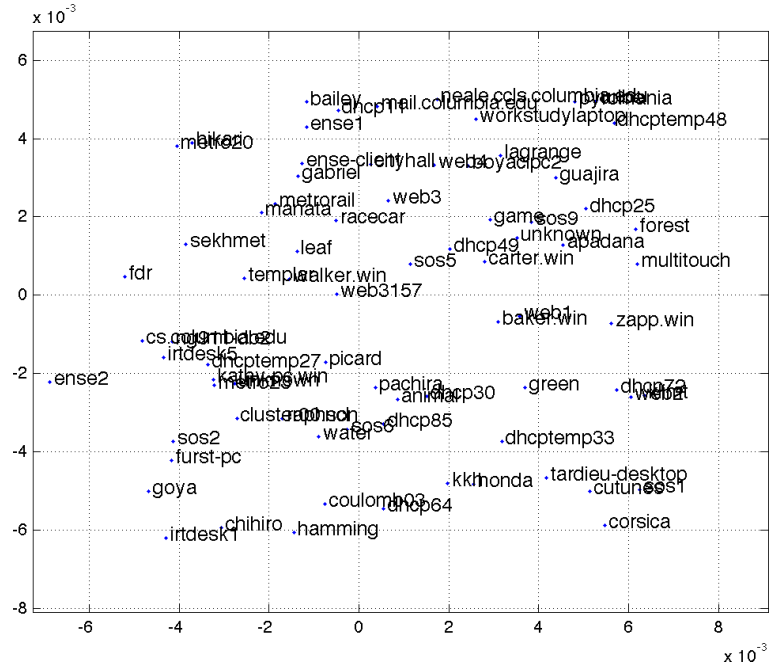
The amount of information preserved is controllable through the use of smoothing transformation described in the Synthesis section. Figure (7.18) shows the embedding of a dataset before and after a "whitening" transformation where we smoothed the Columbia dataset using a large number of nearest neighbors. Visually, we can see the disappearance of recognizable clusters of behavior, as expected.

Top anomalous hosts	Top anomalous clusters
dhcptemp33.cs.columbia.edu	fdr.cs.columbia.edu **dhcptemp33.cs.columbia.edu** picard.cs.columbia.edu klimt.cs.columbia.edu
dhcptemp33.cs.columbia.edu kathy-pc.win.cs.columbia.edu	fdr.cs.columbia.edu dhcptemp33.cs.columbia.edu picard.cs.columbia.edu klimt.cs.columbia.edu web3157.cs.columbia.edu honda.cs.columbia.edu cs.columbia.edu raphson.cs.columbia.edu **kathy-pc.win.cs.columbia.edu**

Table 7.2: Ranked anomalies in original and merged datasets.



(a) Original distribution



(b) Whitened distribution

Figure 7.18: Preservation of statistical information is controllable. (a) shows the embedding of similarly behaving hosts and (b) shows the same dataset after a whitening transform.

7.5.3 Intrusion signatures

Snort signatures are another example of content information, and the preservation of these features is considered in this thesis. Snort is a mostly content-signature-oriented intrusion detection system, which utilizes a set of human-operator-written rules for recognizing potential attacks in web traffic. Previous studies have examined the effects of network-trace obfuscation on Snort performance [Yurcik *et al.*, 2007; Lakkaraju and Slagell, 2008]. Their main results draw two clear conclusions. First, false-positives is the most significant side effect of using this class of traffic-obfuscation techniques. Second, problems occur specifically when port values are obfuscated. Because content is typically not released in network traffic datasets, obfuscation usually has no effect on this aspect of detection – only the header value contents are modified. Given that Snort's detection system selects sets of applicable rules based on observed port values, obfuscating these values can effectively short circuit some detection mechanisms. For example, under the default setting, if a signature exists for an SSH exploit then that signature is only applied when port 22 traffic is observed. This is done to decrease the computational cost of running expensive string-matching algorithms on very large volumes of traffic. The short-circuit effect happens when, during obfuscation, port 22 is converted to another value. This would prevent the SSH-exploit rule from triggering, leading to false negatives. Conversely, introducing false positives is another concern. When obfuscating the header values, certain transformations such as randomizing the port values can have an adverse effect on IDS because these systems look for anomalous traffic to bogus ports as well. Randomly assigning port values to 0, for example, would trigger the Snort "Bad TCP Port Value rule." If signification amounts of port values are changed for a particular host, certain behavior-based detection mechanisms might fail as well, such as rules for port scanning, which assumes a certain distribution of port activity within a window of time.

Given this, we must consider how the behavior-based obfuscation transformations presented in this thesis effects content-based IDS implementations, such as Snort. Fundamentally, our system does not affect these types of signatures for the several key reasons. First, consider the obfuscation techniques used – these are primarily, merging of identities and synthesis of new traffic. In the first case, given that most of the transformation involved

Dataset	1 - 10	11 - 20	21 - 50	50-100	100-200	200-1000	1000+
LBNL Enterprise	20	5	0	1	0	9	0
NSA Capture	23	1	2	0	1	2	16
West Point Border	27	11	0	0	0	1	8
George Mason Univ.	28	26	10	1	0	0	0
Columbia Univ.	50	28	4	0	0	0	0
UMich. Merit	59	34	27	3	2	10	0

Table 7.3: Distribution of model sizes per dataset, in terms of number of port bins used.

in merging traffic resolves around altering the IP addresses, and there are no modifications to port values or the packet content, rules related to these features are not effected by this transformation. Intrusion detection can works as before, with one difference, being that the resulting alerts are attributed to a different IP address than in the original – hindering exact source attribution is not a side-effect, in this case, as source-obfuscation is precisely what anonymization aims to accomplish. Secondly, we consider the effects of using synthesis algorithms to insert new traffic into the dataset. Recall that synthesis within the PTM model is a essentially a Markov random walk on the PTM model parameters. States within the model that correspond to observed port activity in the training dataset are traversed and volumetric information are sampled based on the amount of activity observed in the original traffic. Given this, it is not possible to introduce unobserved port activity in to the dataset because an unobserved state is never reached within the synthesis algorithm. This prevents the introduction of behavior-based anomalies such as the "TCP Port 0" alert Snort, as well as false positives such as mistaken port scanning. Given that the volumetric information is sampled from the distribution trained on the actual data, volumetric behavior is consistent with the original, and false positives such as DDoS attacks would not be introduced.

Table (7.3) shows the sizes of the PTM models in the different datasets. The table shows that the average model size, which relates to the number of ports observed in that host's traffic, is rather small, with most hosts observing less than 10 distinct ports. When merging similar hosts, overlapping port values would mean that the total number of distinct ports introduced in the clustered data does not generate a significant increase. This indicates

that merging different host with distinct ports would not trigger behavior signatures such as port-scanning rules.

Modification of packet content is not handled by our proposed system – it is reasonable to assume that no packet content is released by the provider, and if content were included, then standard obfuscation techniques such as hashing or encryption, such as the primitives offered by the PktAnon system [Gamer *et al.*, 2008], can be used to obfuscate these components independent to our system.

Network communications structure

The network communication structure describes the connectivity of the hosts within the network to each other and the outside world. This feature has studied by modeling the network dispersion graph and identifying hosts based on the pattern of connections. For example, the outbound peering cardinality of each host might be an interesting feature to preserve in the dataset. In addition, graph similarity metrics such as the Jaccard similarity coefficient may be used to represent the dispersion graph structure of a network in a kernel-based learning set up. How to incorporate such graphical structure learning into our current system is discussed in the future works section.

7.6 Connection to existing anonymity metrics

Measuring anonymity is a difficult challenge, especially in the offline data-obfuscation scenario that we study. In this study, we've presented a set of transformation algorithms and demonstrated their accuracy and stability and how they fit together with existing machine learning theory. Further, we demonstrate how to extend our methods in a range of areas including incorporating graphical structure of the network to non-parametric kernel density estimation. As such, our techniques are categorized in the same policy-driven systems such as `tcpmkpub` and `PktAnon`. The techniques presented in this thesis are designed for use in such a policy-driven environment. For example, the optimal k not only depends on policy, but also on the dataset itself. Different behavioral distributions would necessitate different settings for the parameters of our algorithms. In general, unlike specific transformations,

such as a hash function or a prefix-preserving IP address transformation [Fan *et al.*, 2004], it is more difficult to measure the anonymity induced by an entire class of transformations, independent of the data. As an extreme example, if the dataset contained only two hosts and one of them is highly anomalous, then this clustering methodology obviously would not be appropriate, since the anonymity-by-crowds paradigm would not fit this scenario. Fundamentally, anonymity measurement is conditioned on both the data and the policy of the source provider. Any attempt to present a bound on anonymity independent of this would be vacuous. Thus, in this section, we show how to connect our work with existing anonymity metrics, so that different settings of our algorithm may be assigned in an information-theoretic way.

The following anonymity metrics from existing literature are examined and we show how they relate to the algorithms proposed in this thesis.

***k*-anonymity**: Trivially, this is the number of members in a cluster. In the merged traffic dataset, source-attribution is only achievable with $O(1/k)$-precision given that all elements of the cluster behave similarly.

***l*-diversity**: This is a measure of entropy within the elements of a specific measure, typically used in micro-data anonymization analysis, for example, the range of possible values in a particular measurement. For example, a measurement of red-blood cell in a medical record presents a larger variance than an entry that can only contain two possible elements "1" or "0" which might related to whether or not a patient has a certain disease. l-diversity and can be used to derive anonymization policies that restrict the release of certain features that do not exhibit sufficient diversity and could potentially break anonymity. In our setting l-diversity corresponds to the self-similarity measurement presented in § 7.3.1. The diversity in a host's own behavior directly influences the anonymization policy for that host. If the behavior is consistent, for example, then a higher confidence of anonymity may be assigned to any clustering, as opposed to hosts with highly erratic behavior where merging into a crowd does not necessarily guarantee that the unique characteristics of that host does not stand out.

***t*-closeness**: This is a measure of similarity between data elements. Elements with statistical properties that are very close to those of other elements may leak information about each other. For example, consider the range of scores that a committee of reviewers may assign to a paper. One would like to anonymize the score such that it is no possible to determine which member assigned which score. However, if all of the scores were the same, then it is readily apparent what each member's score was. This is the measure referred to as t-closeness. In our environment, it naturally corresponds to the kernel similarity measure between two hosts. The components of the Gram matrix correspond to the individual t-closeness measures of the dataset exactly.

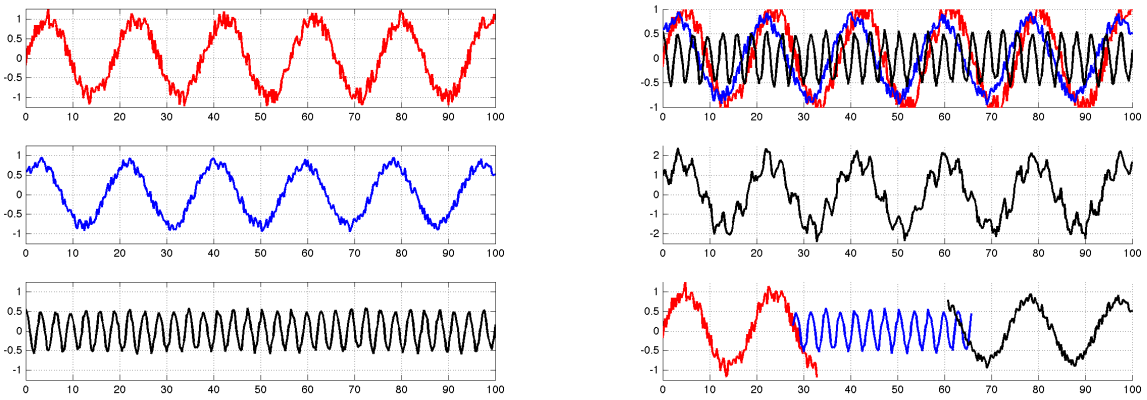
In addition, given that our algorithm is a clustering-based approach, machine learning-theory related to measuring the fidelity of clustering algorithms are directly applicable. The V-Measure technique presented by Rosenberg *et al.* [Rosenberg and Hirschberg, 2007] utilizes a homogeneity measure that analyze the similarity members of a cluster, much in the same way that our parity function analyzes the packet-balance in the resulting merged traffic data. Additional clustering measurement theory are presented by [Meila, 2007]. Given relationship, we bridge the gap between network trace anonymization kernel-based machine learning algorithms and cluster analysis, both of which are well studied disciplines.

7.7 Information leakage and potential attacks

This section discusses some of the technical challenges that is associated with this approach to anonymization. Potential areas of information leakage are discussed and strategies for the mitigation of these effects on privacy preservation are discussed.

7.7.1 On cover-traffic and the need for synchronized overlay

Given that our approach is essentially an offline analogue of the mixnet model for anonymization, the similar timing attacks are applicable. If for example, only one member of a mixnet is active at a given time, then all of the behavioral characteristics of the traffic is fully de-



(a) (b)

Figure 7.19: Visual example of the cover-traffic problem. (a) three similar signals are displayed (b) top plot shows the three signals merged, middle plot shows the aggregate effect, bottom plot shows the problem when traffic sequences are not properly overlaid.

pendent on that particular user. The same is true for the offline traffic merging case. When mixing data, consideration of the traffic window is necessary so that different sessions are indeed overlapping in time. This is the concept of "cover traffic" that has been studied in the past in related anonymization fields. Cover-traffic is commonly used in network anonymity research [Freedman and Morris, 2002] and has been discussed in Tor [Dingledine *et al.*, 2004] and theorized in [Syverson *et al.*, 2001]. Fundamentally, the problem is one of signal averaging.

Consider the examples presented in Figure (7.19). Here, three similar signals, represent different behavior profiles captured with the PTM model (see § 7.3.1). The goal is to merge these so that the characteristics of the traffic are preserved in the aggregate result while, at the same time, the individual signals cannot be easily de-multiplexed. Figure (7.19)(b) shows the acceptable and unacceptable merging results. When the signals are properly aligned with respect to time, the resulting aggregations (in this case represented by the sum of the signals) combines the qualities of all three. However, when the signals are not aligned with respect to time, the signals are easily separable and identifiable as distinct behaviors. This scenario may happen in practice when merging hosts which are not active

```
# wwww:ttt:D:ss:OOO...:QQ:OS:Details
#
# wwww     - window size (can be * or %nnn or Sxx or Txx)
#              "Snn" (multiple of MSS) and "Tnn" (multiple of MTU) are allowed.
# ttt      - initial TTL
# D        - don't fragment bit (0 - not set, 1 - set)
# ss       - overall SYN packet size (* has a special meaning)
# OOO      - option value and order specification (see below)
# QQ       - quirks list (see below)
# OS       - OS genre (Linux, Solaris, Windows)
# details  - OS description (2.0.27 on x86, etc)
```

Figure 7.20: p0f signature format for passive OS fingerprinting.

at the same time. Thus, it is necessary to take into account the time-window of the traffic when merging. In practice, is best to change the time-stamps of the sessions in order to ensure that the resulting traffic is properly overlapping.

7.7.2 Network device fingerprints

Network devices not only exhibit statistical fingerprints, but also content fingerprints as well. Packet-payloads not considered, the packet header information present discriminative information about the network device that generated the traffic stream. The field of passive operating system fingerprinting studies this property. The most well known tool in this area is `p0f`.

p0f identifies operating systems based on the following features shown in Figure (7.20). When negotiating a TCP connection the initiating host sends a SYN packet with information about itself. This information consists of the initial window size, the initial time-to-live value (TTL), and maximum segment size (MSS). These values are based on the implementation of the TCP stack and are distinct – to a certain degree — among operating systems, as well as different versions of the same operating system. In addition, characteristics such as the size of the initial SYN packet, the presence of the do-not-fragment flag, are features which can be used to determine the type and version of the operating system. Not only are

operating systems identifiable in this way, but also network devices such as routers, mobile phones, and any other device which and send and receive TCP data.

In practice, the discriminative power of these features has been found to be only somewhat reliable. Many instances of Windows, for example, use a predictable initial window size of 65,536 and a TTL of 128. An up-to-date list of the p0f signatures are presented in the appendix. In our experiments, we've found that p0f is only able to recognize devices with accuracy around 15%, mostly in identifying Windows from Linux machines. Though the discriminative power of these signatures is not high, care should taken that these signatures are kept consistent in the anonymized dataset. When assigning group identities, the passive OS fingerprint of the cluster can be assigned in the same manner as the IP address. This can be done by selecting one member of the group to represent the rest: the IP address of that member is assigned to all members of the group, and the corresponding identifying information such as Ethernet header values and passive OS fingers should be carried over to the rest of the group as well.

7.7.3 Active attacks and artifact insertion

Active attacks attempt to embed fingerprints into the dataset prior to anonymization so that they may be recoverable in the transformed data and used for de-anonymization. This can include the insertion of specially crafted packets with embedded signatures in the header values, sending traffic to a host on a set of specific ports in a predetermined order ("port knocking") or embedding port scans in the traffic so that prefix-preserved subnets IP addresses may be identified, as described in the background chapter. These attacks do not effect the assumption about anonymity described in the first section of this chapter. Given these artifact embeddings, the attacker can only recover the pseudonymous identify of the target, he cannot infer anything about the other traffic sessions attributed to that pseudonym. The mixing factor is not conditioned on network subnet topology, but rather on behavior, thus port-scanning features are obfuscated by nature. Port-knocking signatures, like content signatures, identify only the pseudonym for the target.

Concluding remarks

Traffic anonymization is, at its core, a moving challenging; one whose parameters change with the times as new techniques in emerges. Information that has always existed in the traffic are extracted in new ways and learned exploited to derive information previously unknown. The IP ID field was not a source of information leakage until it was used to count hosts behind a NAT, neither was packet sizes in anonymized netflow data considered sensitive until it was shown that the distribution of such sizes form a discriminative profile for the websites visited. Such are the challenges facing network trace anonymization that make it unique and separate from similar studies in microdata. It has to be reiterated that trace anonymization is fundamentally risk-management. The techniques we proposed add a new dimension to trace anonymization and raise the bar for the attacker.

Chapter 8

Limitations, extensions, and future work

Introduction

This chapter explains the current imitations of our work, how these limitations are to be addressed, and describes several potential directions in which this work can be expanded. These extensions include incorporating the communications structure of the network in the clustering function, and anonymization approaches involving exchange models instead of anonymized data.

8.1 Limitations of the behavior-based approach

This section enumerates the different types of limitations in anonymization approach. These include both algorithmic and conceptual limitations.

TCP-only: The behavior model is conditioned entirely on the TCP traffic generated by host. Therefore any privacy-piercing signatures that may exist in the other protocols may be overlooked. Anonymizing the data would require dropping all non-TCP traffic in order to minimize the chances of potential side-channel information leakage. This is an undesired downside of the current system. Ideally, the other protocols would factor into the behavior model as well, or are dealt with in a principled manner consistent with this mixnet approach. However, given that this is a first step in this direction for network trace anonymization research, the focus was restricted only to TCP. In future work, other protocols should be examined.

Time information: As mentioned in the synthesis section, the PTM does not track the timestamp feature of the data. Therefore, behavior is only track with respect to transition between each state, and not conditioned on time. Behavior signatures that are time-dependent could possibly exist in the dataset. For example, if a certain host generates traffic on a specific port with a certain frequency during the day. It might be possible to isolate this signal from the mixed dataset post-anonymization, depending on the significance of the signal-to-noise ratio. This challenging can be addressed by incorporating timing information in the models in future work. Modeling the inter-arrival time between network sessions

is also an important feature that is not yet studied within this thesis. This behavior can be modeled as an exponential distribution with expected inter-arrivals determined by Poisson processes.

Scope of behavior: The model described in this thesis measures the aggregate outbound traffic profile for a host for the behavior profile. As such it treats all such outbound connections the same, regardless of destination. A stronger measurement might take into account the end points of each connection so that distinctions can be made based on the type of connection. Further, incoming traffic profiles are not directly used in the model. This is not exactly an oversight, as our experiment were performed over incoming traffic behavior as well; these models generally performed worse as they added more noise to the models. A host cannot control the type of traffic that it receives; all hosts that are port scanned in a certain manner may appear similar to each other, for example. How to properly model incoming traffic and make use of this signal is the subject of future work.

Communications structure: The current system is agnostic to the underlying network topology. However, this information may prove useful for similarity measurements between hosts. For example, while two hosts may exercise a different set of services and protocols, if they have the same set of peers within the network then this should factor into the measurement for similarity. This particular point is discussed in further detail in the following section.

Content: No packet payload information is used in the behavior profiles. This is a limitation, though it is not a very restrictive one, given that few source provide un-obfuscated content in the data that they release. However, packet header content is available and is an under-utilized feature in our models. While we do account for passive OS fingerprints, there are many ways to make further use of these signatures, such as driving the cluster by first labeling the host with the OS label, and ensuring that hosts are only clustered with peers running the same OS. Conceptually, this is fairly straightforward, and can be explored in future work.

8.2 Incorporating the network topology in the feature set

One of the more promising directions with which we can extend this work is making use of network topology information in the behavior modeling. Host-role prediction based solely on network topology has been studied in the past [Tan *et al.*, 2003]. In the following subsections, we describe possible ways to incorporate this information.

8.2.1 The traffic-dispersion graph

The traffic dispersion graph measure the connection peering of hosts on a network. This indicates which machines connect to which other ones, and it is possible to derive some notion of role from this information.

Consider the figure (8.1). This is a small-world-graph representation of the traffic dispersion among a group of Columbia hosts – notice that multiple hosts connect to the "int-ns1" node. Not surprisingly, this is our department's primary DNS server. One may wonder why DNS connections are observable over TCP, this is because DNS results over 1500 bytes in length are sent over TCP instead of the default UDP. This is one example of how roles may be easily identifiable based on the traffic communications pattern. Behaviorally, there may or may not be any significantly discriminative signals from int-ns1, or alternatively if there is a host with very little traffic but the traffic that it does emit follow a pattern very similar to a DNS server, such information would be interesting to identify from the data.

8.2.2 Features from structure

Several potential graph-based features exists, these include the outbound-cardinality, which involves measuring the number of external hosts any particular internal host connects to as an element of behavior profiling. Looking at similarity as a function of internal all-pairs shortest paths might help infer some structure information about the subnets and which hosts act as servers and which are user machines. A related measurement is the Jaccard similarity coefficient that measure similarity as a function of the size of the set of commons peers within multiple hops on the graph.

Figure (8.2) shows a the outbound cardinality distribution. This shows the distribution

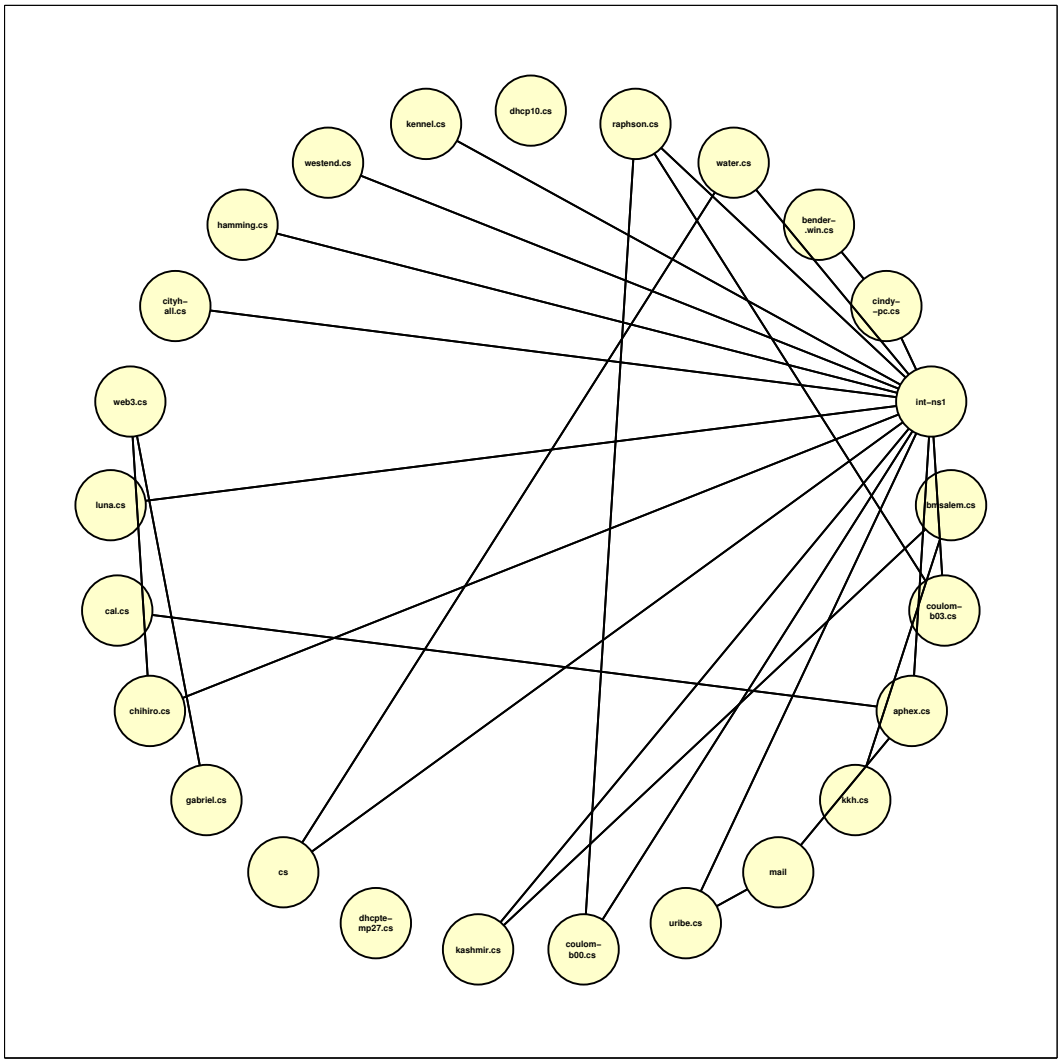


Figure 8.1: Small-world-graph connection patterns for a set of hosts on the Columbia network.

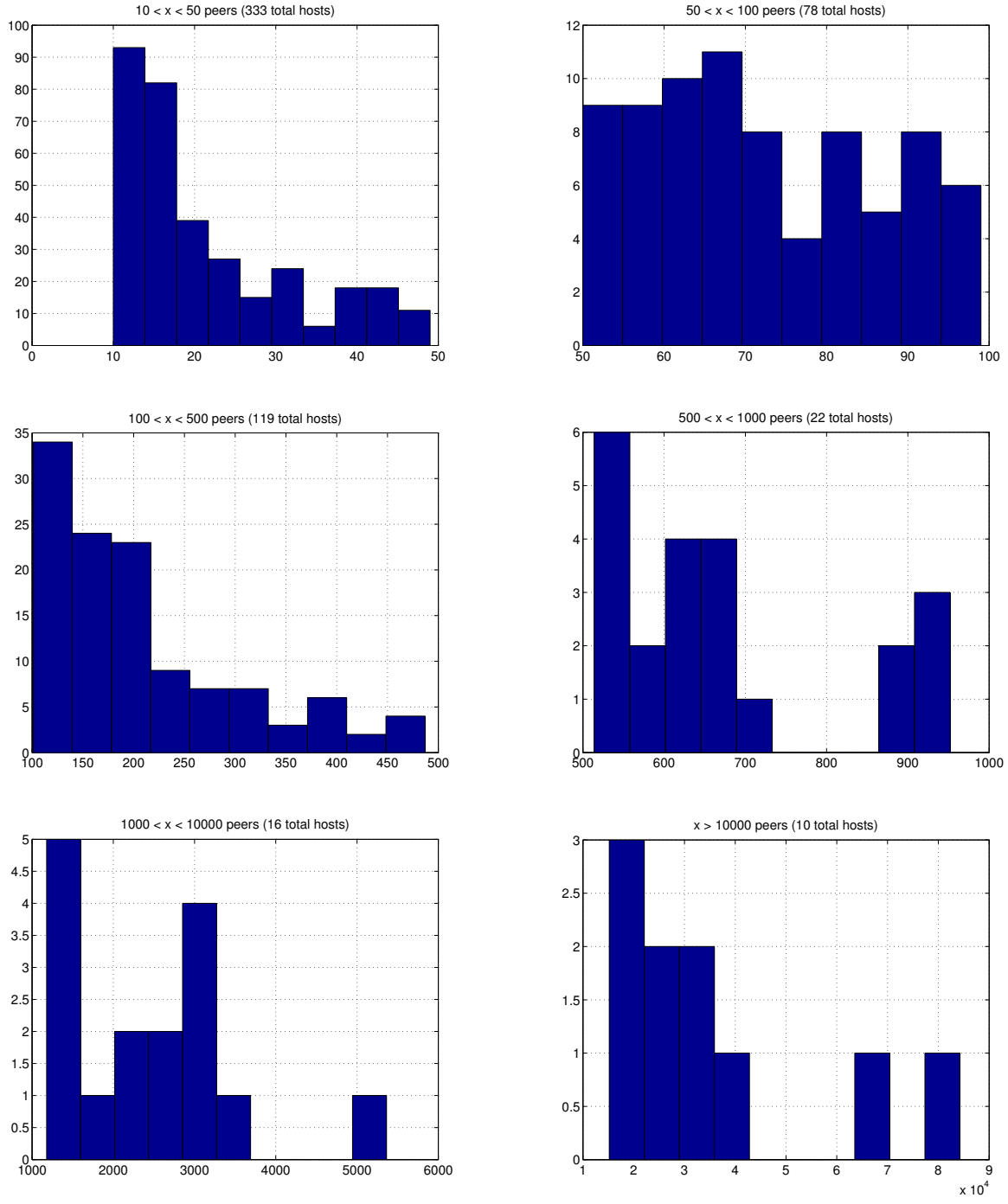


Figure 8.2: Peer cardinality histograms. 1000 hosts on the CU network, sampled over 4 hours across 20 days.

of hosts based on the number of distinct peers they connected with. As the figures show, there is a rich distribution of hosts based on this feature. Without looking at any additional content it is obvious which hosts are web servers as these are the only servers that can possibly maintain over 80,000 peers. Clustering hosts based on this information should be explored in future work.

8.2.3 Structure-based host clustering

As discussed previously, there are many potential sets of features recoverable from the network's dispersion graph. Among these are the peer cardinality and the Jaccard coefficient. The best way to balance these features with the behavior measurement already examined is also a matter of future work. Given that our work is based on kernel methods, the immediate intuition is to draw insight from the field of kernel learning. Kernel learning, within machine learning, is precisely the study of how to combine multiple similarity metrics in order to improve performance. Some immediately related works in is area include learning with graph kernels [Vishwanathan *et al.*, 2010] and multiple kernel learning [Gönen and Alpaydin, 2011].

8.3 Anonymity via model-exchange

In this thesis, we have demonstrated the ability to represent traffic behavior using time-series models, with favorable accuracy. Also demonstrated is the approach to reverse direction, of synthesizing packet trace data using these models. Given this, a few potential directions of research based on model-exchange is immediately available.

Model-comparison and re-synthesis: For certain research purposes only a measurement of similarity in behavior is necessary. This includes intrusion detection, detection of worm and bot traffic, and other security-related fields. Given this, there might not be a need to explicitly compare the actual data samples. The PTM already tracks the unique ports observed within the data, thus a range of behavior-based signatures is already representable. The NSA dataset for example, contain several instances of port-scanning behavior, which

were all clustered together in the results in our studies. The ability to cluster malicious traffic, such as those belonging to worms, bots, and other infected machines, should be examined in future work to see if this type of behavior model can be exchanged for security/forensic research purposes without loss of privacy.

Behavior matching with exchanged models: Query-based system for network trace sharing has already been explored in the past, such as the SC2D system [Mogul and Arlitt, 2006]. These systems are not widely used because strict sets of quantifiers for network characteristics are not easily enumerable. In a related field, microdata anonymization based on differential privacy is growing to be a well-understood theory. DP measures privacy by deriving bounds based on type of queries that may be submitted to the statistical dataset. Given this, it is natural to consider the possibility of a behavior-based query system for network traces. Instead of exchanging traffic, or behavior models, the querying party trains a model on a segment of traffic that they possess. They can then submit this model to a query system that would reply with whether or not this behavior pattern was observed in their own system. In fact, the spectral partition function can be adapted to be used as a binary-search algorithm for fast query matching. This might be a potential method with which organization can securely determine the existing of worms and bots on their networks. Further, results from differential privacy might be applicable in this setting, thus this would be a potential direction that allows differential privacy to be applied to network trace anonymization.

8.4 Behavior-driven anonymous VPN

The behavior-based approach studied for offline data anonymization might be extended to the online setting if we train models for user behavior, group these users based on similarity, then route their traffic such that traffic belonging to members of the same cluster exits from the same nodes in an anonymous VPN system. This is essentially adding a behavior-based circuit construction layer on top of a Tor-like framework. Most enterprise environments already maintain VPN systems, changing the routing policy to concentrate traffic based on

behavior might be a simple extension, and one which can provide a higher level of anonymity in an online setting. Such techniques might be effective in hindering timing-based attacks used against current anonymous VPN systems. This is another avenue for future work.

Chapter 9

Conclusions

This thesis proposes a new approach to behavior-based, statistics-preserving, network trace source-anonymization, one that is based on novel measurements of host behavior and similarity, where anonymity and statistics-preservation are congruent objectives in an unsupervised-learning problem. This approach is compatible with existing anonymization frameworks, and adds a new layer to our understanding of network trace anonymization.

This thesis presented a new and efficient time-series model for host behavior, trained using high-level flow-layer meta-data, whose training and likelihood-evaluation times maintain linear growth in the number of data samples. Evaluation show the accuracy of this model in recognizing distinct classes of network behavior. A new kernel, motivated by the concept of probability product kernels, was derived for comparing similarity between models in an efficient and scalable manner. This proposed kernel is efficient in both runtime and memory costs, requiring an order of magnitude less computation time than similar methods, while maintaining high classification accuracy. New graph-partition-motivated clustering methods, were proposed which utilizes these models and kernels. We showed dramatic improvements in both accuracy and runtime over existing methods. New methods for network behavior visualization were also proposed, based on these techniques. Algorithms for synthesizing network traffic according to targeted behavior, were also provided, with evaluation results that demonstrate measurable fidelity. Behavior-interpolation, regression, and shaping, as well as the techniques for translating statistical samples back into packet trace data were explored. The techniques described in this thesis forms a new framework for behavior-based network trace anonymization. We demonstrated the stability and accuracy of this approach. Connections to existing anonymization theory are shown and we demonstrate techniques for measuring anonymization requirements from the data with empirical results over a range of real traffic datasets.

We showed that source-anonymity is directly related to the size of the anonymity group, the homogeneity in the behavior of the individual members, and metrics for these qualities are derived. A machine-learning-motivated framework is proposed with which we can measure behavior with quantifiable accuracy, and the optimal segmentation of the population into anonymized groupings is shown to be recoverable under a graph-partitioning problem, where maximization of cluster-homogeneity is an intrinsic property of the solution.

Algorithms that guarantee a minimum anonymity set-size are presented, as well as novel techniques for behavior visualization and information compression. Empirical results on a range of network traffic datasets demonstrate the superior performance of our methods over similar techniques in accuracy, space efficiency, and runtime.

Part I

Appendices

.1 p0f signature list

```
#
# p0f - SYN fingerprints
# ----------------------

##########################
# Standard OS signatures #
##########################

# ----------------- Linux -------------------

S1:64:0:44:M*:A:Linux:1.2.x
512:64:0:44:M*:.:Linux:2.0.3x (1)
16384:64:0:44:M*:.:Linux:2.0.3x (2)

# Endian snafu! Nelson says "ha-ha":
2:64:0:44:M*:.:Linux:2.0.3x (MkLinux) on Mac (1)
64:64:0:44:M*:.:Linux:2.0.3x (MkLinux) on Mac (2)

S4:64:1:60:M1360,S,T,N,W0:.:Linux:2.4 (Google crawlbot)
S4:64:1:60:M1430,S,T,N,W0:.:Linux:2.4-2.6 (Google crawlbot)

S2:64:1:60:M*,S,T,N,W0:.:Linux:2.4 (large MTU?)
S3:64:1:60:M*,S,T,N,W0:.:Linux:2.4 (newer)
S4:64:1:60:M*,S,T,N,W0:.:Linux:2.4-2.6

S3:64:1:60:M*,S,T,N,W1:.:Linux:2.6, seldom 2.4 (older, 1)
S4:64:1:60:M*,S,T,N,W1:.:Linux:2.6, seldom 2.4 (older, 2)
S3:64:1:60:M*,S,T,N,W2:.:Linux:2.6, seldom 2.4 (older, 3)
S4:64:1:60:M*,S,T,N,W2:.:Linux:2.6, seldom 2.4 (older, 4)
T4:64:1:60:M*,S,T,N,W2:.:Linux:2.6 (older, 5)

S4:64:1:60:M*,S,T,N,W5:.:Linux:2.6 (newer, 1)
S4:64:1:60:M*,S,T,N,W6:.:Linux:2.6 (newer, 2)
S4:64:1:60:M*,S,T,N,W7:.:Linux:2.6 (newer, 3)
T4:64:1:60:M*,S,T,N,W7:.:Linux:2.6 (newer, 4)


S20:64:1:60:M*,S,T,N,W0:.:Linux:2.2 (1)
S22:64:1:60:M*,S,T,N,W0:.:Linux:2.2 (2)
S11:64:1:60:M*,S,T,N,W0:.:Linux:2.2 (3)
```

```
# Popular cluster config scripts disable timestamps and
# selective ACK:

S4:64:1:48:M1460,N,W0:.:Linux:2.4 in cluster

# This happens only over loopback, but let's make folks happy:
32767:64:1:60:M16396,S,T,N,W0:.:Linux:2.4 (loopback)
32767:64:1:60:M16396,S,T,N,W2:.:Linux:2.6 (newer, loopback)
S8:64:1:60:M3884,S,T,N,W0:.:Linux:2.2 (loopback)

# Opera visitors:
16384:64:1:60:M*,S,T,N,W0:.:-Linux:2.2 (Opera?)
32767:64:1:60:M*,S,T,N,W0:.:-Linux:2.4 (Opera?)

# Some fairly common mods & oddities:
S22:64:1:52:M*,N,N,S,N,W0:.:Linux:2.2 (tstamp-)
S4:64:1:52:M*,N,N,S,N,W0:.:Linux:2.4 (tstamp-)
S4:64:1:52:M*,N,N,S,N,W2:.:Linux:2.6 (tstamp-)
S4:64:1:44:M*:.:Linux:2.6? (barebone, rare!)
T4:64:1:60:M1412,S,T,N,W0:.:Linux:2.4 (rare!)

# ----------------- FreeBSD -----------------

16384:64:1:44:M*:.:FreeBSD:2.0-4.2
16384:64:1:60:M*,N,W0,N,N,T:.:FreeBSD:4.4 (1)

1024:64:1:60:M*,N,W0,N,N,T:.:FreeBSD:4.4 (2)

57344:64:1:44:M*:.:FreeBSD:4.6-4.8 (RFC1323-)
57344:64:1:60:M*,N,W0,N,N,T:.:FreeBSD:4.6-4.9

32768:64:1:60:M*,N,W0,N,N,T:.:FreeBSD:4.8-5.1 (or MacOS X 10.2-10.3)
65535:64:1:60:M*,N,W0,N,N,T:.:FreeBSD:4.7-5.2 (or MacOS X 10.2-10.4) (1)
65535:64:1:60:M*,N,W1,N,N,T:.:FreeBSD:4.7-5.2 (or MacOS X 10.2-10.4) (2)

65535:64:1:60:M*,N,W0,N,N,T:Z:FreeBSD:5.1 (1)
65535:64:1:60:M*,N,W1,N,N,T:Z:FreeBSD:5.1 (2)
65535:64:1:60:M*,N,W2,N,N,T:Z:FreeBSD:5.1 (3)
65535:64:1:64:M*,N,N,S,N,W1,N,N,T:.:FreeBSD:5.3-5.4
65535:64:1:64:M*,N,W1,N,N,T,S,E:P:FreeBSD:6.x (1)
65535:64:1:64:M*,N,W0,N,N,T,S,E:P:FreeBSD:6.x (2)

65535:64:1:44:M*:Z:FreeBSD:5.2 (RFC1323-)
```

```
# 16384:64:1:60:M*,N,N,N,N,N,N,T:.:FreeBSD:4.4 (tstamp-)

# ----------------- NetBSD ------------------

16384:64:0:60:M*,N,W0,N,N,T:.:NetBSD:1.3
65535:64:0:60:M*,N,W0,N,N,T0:.:-NetBSD:1.6 (Opera)
16384:64:1:60:M*,N,W0,N,N,T0:.:NetBSD:1.6
65535:64:1:60:M*,N,W1,N,N,T0:.:NetBSD:1.6W-current (DF)
65535:64:1:60:M*,N,W0,N,N,T0:.:NetBSD:1.6X (DF)
32768:64:1:60:M*,N,W0,N,N,T0:.:NetBSD:1.6Z or 2.0 (DF)
32768:64:1:64:M1416,N,W0,S,N,N,N,N,T0:.:NetBSD:2.0G (DF)
32768:64:1:64:M*,N,W0,S,N,N,N,N,T0:.:NetBSD:3.0 (DF)

# ----------------- OpenBSD -----------------

16384:64:1:64:M*,N,N,S,N,W0,N,N,T:.:OpenBSD:3.0-3.9
57344:64:1:64:M*,N,N,S,N,W0,N,N,T:.:OpenBSD:3.3-3.4
16384:64:0:64:M*,N,N,S,N,W0,N,N,T:.:OpenBSD:3.0-3.4 (scrub)
65535:64:1:64:M*,N,N,S,N,W0,N,N,T:.:-OpenBSD:3.0-3.4 (Opera?)
32768:64:1:64:M*,N,N,S,N,W0,N,N,T:.:OpenBSD:3.7

# ----------------- Solaris -----------------

S17:64:1:64:N,W3,N,N,T0,N,N,S,M*:.:Solaris:8 (RFC1323 on)
S17:64:1:48:N,N,S,M*:.:Solaris:8 (1)
S17:255:1:44:M*:.:Solaris:2.5-7 (1)

# Sometimes, just sometimes, Solaris feels like coming up with
# rather arbitrary MSS values ;-)

S6:255:1:44:M*:.:Solaris:2.5-7 (2)
S23:64:1:48:N,N,S,M*:.:Solaris:8 (2)
S34:64:1:48:M*,N,N,S:.:Solaris:9
S34:64:1:48:M*,N,N,N,N:.:Solaris:9 (no sack)
S44:255:1:44:M*:.:Solaris:7

4096:64:0:44:M1460:.:SunOS:4.1.x

S34:64:1:52:M*,N,W0,N,N,S:.:Solaris:10 (beta)
32850:64:1:64:M*,N,N,T,N,W1,N,N,S:.:Solaris:10 (1203?)
32850:64:1:64:M*,N,W1,N,N,T,N,N,S:.:Solaris:9.1
```

```
# ----------------- MacOS -------------------

S2:255:1:48:M*,W0,E:.:MacOS:8.6 classic

16616:255:1:48:M*,W0,E:.:MacOS:7.3-8.6 (OTTCP)
16616:255:1:48:M*,N,N,N,E:.:MacOS:8.1-8.6 (OTTCP)
32768:255:1:48:M*,W0,N:.:MacOS:9.0-9.2

32768:255:1:48:M1380,N,N,N,N:.:MacOS:9.1 (OT 2.7.4) (1)
65535:255:1:48:M*,N,N,N,N:.:MacOS:9.1 (OT 2.7.4) (2)

65535:64:1:64:M1460,N,W3,N,N,T,S,E:PI:MacOS:Snow Leopard

# ----------------- Windows -----------------

# Windows TCP/IP stack is a mess. For most recent XP, 2000 and
# even 98, the pathlevel, not the actual OS version, is more
# relevant to the signature. They share the same code, so it would
# seem. Luckily for us, almost all Windows 9x boxes have an
# awkward MSS of 536, which I use to tell one from another
# in most difficult cases.

8192:32:1:44:M*:.:Windows:3.11 (Tucows)
S44:64:1:64:M*,N,W0,N,N,T0,N,N,S:.:Windows:95
8192:128:1:64:M*,N,W0,N,N,T0,N,N,S:.:Windows:95b

# There were so many tweaking tools and so many stack versions for
# Windows 98 it is no longer possible to tell them from each other
# without some very serious research. Until then, there's an insane
# number of signatures, for your amusement:

S44:32:1:48:M*,N,N,S:.:Windows:98 (low TTL) (1)
8192:32:1:48:M*,N,N,S:.:Windows:98 (low TTL) (2)
%8192:64:1:48:M536,N,N,S:.:Windows:98 (13)
%8192:128:1:48:M536,N,N,S:.:Windows:98 (15)
S4:64:1:48:M*,N,N,S:.:Windows:98 (1)
S6:64:1:48:M*,N,N,S:.:Windows:98 (2)
S12:64:1:48:M*,N,N,S:.:Windows:98 (3
T30:64:1:64:M1460,N,W0,N,N,T0,N,N,S:.:Windows:98 (16)
32767:64:1:48:M*,N,N,S:.:Windows:98 (4)
37300:64:1:48:M*,N,N,S:.:Windows:98 (5)
46080:64:1:52:M*,N,W3,N,N,S:.:Windows:98 (RFC1323+)
65535:64:1:44:M*:.:Windows:98 (no sack)
```

```
S16:128:1:48:M*,N,N,S:.:Windows:98 (6)
S16:128:1:64:M*,N,W0,N,N,T0,N,N,S:.:Windows:98 (7)
S26:128:1:48:M*,N,N,S:.:Windows:98 (8)
T30:128:1:48:M*,N,N,S:.:Windows:98 (9)
32767:128:1:52:M*,N,W0,N,N,S:.:Windows:98 (10)
60352:128:1:48:M*,N,N,S:.:Windows:98 (11)
60352:128:1:64:M*,N,W2,N,N,T0,N,N,S:.:Windows:98 (12)

# What's with 1414 on NT?
T31:128:1:44:M1414:.:Windows:NT 4.0 SP6a (1)
64512:128:1:44:M1414:.:Windows:NT 4.0 SP6a (2)
8192:128:1:44:M*:.:Windows:NT 4.0 (older)

# Windows XP and 2000. Most of the signatures that were
# either dubious or non-specific (no service pack data)
# were deleted and replaced with generics at the end.

65535:128:1:48:M*,N,N,S:.:Windows:2000 SP4, XP SP1+
%8192:128:1:48:M*,N,N,S:.:Windows:2000 SP2+, XP SP1+ (seldom 98)
S20:128:1:48:M*,N,N,S:.:Windows:SP3
S45:128:1:48:M*,N,N,S:.:Windows:2000 SP4, XP SP1+ (2)
40320:128:1:48:M*,N,N,S:.:Windows:2000 SP4

S6:128:1:48:M*,N,N,S:.:Windows:XP, 2000 SP2+
S12:128:1:48:M*,N,N,S:.:Windows:XP SP1+ (1)
S44:128:1:48:M*,N,N,S:.:Windows:XP SP1+, 2000 SP3
64512:128:1:48:M*,N,N,S:.:Windows:XP SP1+, 2000 SP3 (2)
32767:128:1:48:M*,N,N,S:.:Windows:XP SP1+, 2000 SP4 (3)

# Windows 2003 & Vista

8192:128:1:52:M*,W8,N,N,N,S:.:Windows:Vista (beta)
32768:32:1:52:M1460,N,W0,N,N,S:.:Windows:2003 AS
65535:64:1:52:M1460,N,W2,N,N,S:.:Windows:2003 (1)
65535:64:1:48:M1460,N,N,S:.:Windows:2003 (2)

# Odds, ends, mods:

S52:128:1:48:M1260,N,N,S:.:Windows:XP/2000 via Cisco
65520:128:1:48:M*,N,N,S:.:Windows:XP bare-bone
16384:128:1:52:M536,N,W0,N,N,S:.:Windows:2000 w/ZoneAlarm?
2048:255:0:40:.:.:Windows:.NET Enterprise Server
44620:64:0:48:M*,N,N,S:.:Windows:ME no SP (?)
```

```
S6:255:1:48:M536,N,N,S:.:Windows:95 winsock 2
32000:128:0:48:M*,N,N,S:.:Windows:XP w/Winroute?
16384:64:1:48:M1452,N,N,S:.:Windows:XP w/Sygate? (1)
17256:64:1:48:M1460,N,N,S:.:Windows:XP w/Sygate? (2)

# No need to be more specific, it passes:
*:128:1:48:M*,N,N,S:U:-Windows:XP/2000 while downloading (leak!)

# ----------------- BSD/OS ------------------

8192:64:1:60:M1460,N,W0,N,N,T:.:BSD/OS:3.1-4.3 (or MacOS X 10.2)

##########################################
# Appliance / embedded / other signatures #
##########################################

# ---------- Firewalls / routers ------------

S12:64:1:44:M1460:.:@Checkpoint:(unknown 1)
S12:64:1:48:N,N,S,M1460:.:@Checkpoint:(unknown 2)
4096:32:0:44:M1460:.:ExtremeWare:4.x

S32:64:0:68:M512,N,W0,N,N,T,N,N,?12:.:Nokia:IPSO w/Checkpoint NG FP3
S16:64:0:68:M1024,N,W0,N,N,T,N,N,?12:.:Nokia:IPSO 3.7 build 026

S4:64:1:60:W0,N,S,T,M1460:.:FortiNet:FortiGate 50

8192:64:1:44:M1460:.:@Eagle:Secure Gateway

# ------- Switches and other stuff ----------

4128:255:0:44:M*:Z:Cisco:7200, Catalyst 3500, etc
S8:255:0:44:M*:.:Cisco:12008
S4:255:0:44:M536:Z:Cisco:IOS 11.0
60352:128:1:64:M1460,N,W2,N,N,T,N,N,S:.:Alteon:ACEswitch
64512:128:1:44:M1370:.:Nortel:Contivity Client

# ---------- Caches and whatnots ------------

8190:255:0:44:M1428:.:Google:Wireless Transcoder (1)
8190:255:0:44:M1460:.:Google:Wireless Transcoder (2)
8192:64:1:64:M1460,N,N,S,N,W0,N,N,T:.:NetCache:5.2
16384:64:1:64:M1460,N,N,S,N,W0,N:.:NetCache:5.3
```

```
65535:64:1:64:M1460,N,N,S,N,W*,N,N,T:.:NetCache:5.3-5.5 (or FreeBSD 5.4)
20480:64:1:64:M1460,N,N,S,N,W0,N,N,T:.:NetCache:4.1
S44:64:1:64:M1460,N,N,S,N,W0,N,N,T:.:NetCache:5.5

32850:64:1:64:N,W1,N,N,T,N,N,S,M*:.:NetCache:Data OnTap 5.x

65535:64:0:60:M1460,N,W0,N,N,T:.:CacheFlow:CacheOS 4.1
8192:64:0:60:M1380,N,N,N,N,N,N,T:.:CacheFlow:CacheOS 1.1

S4:64:0:48:M1460,N,N,S:.:Cisco:Content Engine

27085:128:0:40:.:.:Dell:PowerApp cache (Linux-based)

65535:255:1:48:N,W1,M1460:.:Inktomi:crawler
S1:255:1:60:M1460,S,T,N,W0:.:LookSmart:ZyBorg

16384:255:0:40:.:.:Proxyblocker:(what's this?)

65535:255:0:48:M*,N,N,S:.:Redline: T|X 2200

# ----------- Embedded systems --------------

S9:255:0:44:M536:.:PalmOS:Tungsten T3/C
S5:255:0:44:M536:.:PalmOS:3/4
S4:255:0:44:M536:.:PalmOS:3.5
2948:255:0:44:M536:.:PalmOS:3.5.3 (Handera)
S29:255:0:44:M536:.:PalmOS:5.0
16384:255:0:44:M1398:.:PalmOS:5.2 (Clie)
S14:255:0:44:M1350:.:PalmOS:5.2.1 (Treo)
16384:255:0:44:M1400:.:PalmOS:5.2 (Sony)

S23:64:1:64:N,W1,N,N,T,N,N,S,M1460:.:SymbianOS:7
8192:255:0:44:M1460:.:SymbianOS:6048 (Nokia 7650?)
8192:255:0:44:M536:.:SymbianOS:(Nokia 9210?)
S22:64:1:56:M1460,T,S:.:SymbianOS:? (SE P800?)
S36:64:1:56:M1360,T,S:.:SymbianOS:60xx (Nokia 6600?)
S36:64:1:60:M1360,T,S,W0,E:.:SymbianOS:60xx

32768:32:1:44:M1460:.:Windows:CE 3

# Perhaps S4?
5840:64:1:60:M1452,S,T,N,W1:.:Zaurus:3.10
```

```
32768:128:1:64:M1460,N,W0,N,N,T0,N,N,S:.:PocketPC:2002

S1:255:0:44:M346:.:Contiki:1.1-rc0

4096:128:0:44:M1460:.:Sega:Dreamcast Dreamkey 3.0
T5:64:0:44:M536:.:Sega:Dreamcast HKT-3020 (browser disc 51027)
S22:64:1:44:M1460:.:Sony:Playstation 2 (SOCOM?)

S12:64:0:44:M1452:.:AXIS:Printer Server 5600 v5.64

3100:32:1:44:M1460:.:Windows:CE 2.0

####################################
# Generic signatures - just in case #
####################################

*:128:1:52:M*,N,W0,N,N,S:.:@Windows:XP/2000 (RFC1323+, w, tstamp-)
*:128:1:52:M*,N,W*,N,N,S:.:@Windows:XP/2000 (RFC1323+, w+, tstamp-)
*:128:1:52:M*,N,N,T0,N,N,S:.:@Windows:XP/2000 (RFC1323+, w-, tstamp+)
*:128:1:64:M*,N,W0,N,N,T0,N,N,S:.:@Windows:XP/2000 (RFC1323+, w, tstamp+)
*:128:1:64:M*,N,W*,N,N,T0,N,N,S:.:@Windows:XP/2000 (RFC1323+, w+, tstamp+)

*:128:1:48:M536,N,N,S:.:@Windows:98
*:128:1:48:M*,N,N,S:.:@Windows:XP/2000
```

Part II

Bibliography

Bibliography

[Alllman and Paxson, 2007] Mark Alllman and Vern Paxson. Issues and etiquette concerning use of shared measurement data. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, IMC '07, pages 135–140, New York, NY, USA, 2007. ACM.

[Allman *et al.*, 2005] Mark Allman, Mike Bennett, Martin Casado, Scott Crosby, Jason Lee, Boris Nechaev, Ruoming Pang, Vern Paxson, and Brian Tierney. LBNL/ICSI Enterprise Tracing Project, 2005.

[Bejtlich *et al.*, 2011] Richard Bejtlich, JJ Cummings, and Sharri Parsell. Openpacket.org: a centralized repository of network traffic traces for researchers, 2011.

[Bellovin, 2002] Steven M. Bellovin. A technique for counting NATted hosts. In *Proc. Second Internet Measurement Workshop*, pages 267–272, Marseille, 2002.

[Biondi, 2012] Philippe Biondi. Scapy - a packet manipulation library, 2012.

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

[BreakingPoint Systems, 2011] BreakingPoint Systems. Breakingpoint network traffic generation systems, 2011.

[Brickell and Shmatikov, 2008] Justin Brickell and Vitaly Shmatikov. The cost of privacy: destruction of data-mining utility in anonymized data publishing. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '08, pages 70–78, New York, NY, USA, 2008. ACM.

[Brugger, 2007] Terry Brugger. KDD Cup '99 dataset (Network Intrusion) considered harmful. http://www.kdnuggets.com/news/2007/n18/4i.html, September 2007.

[Burkhart *et al.*, 2008] Martin Burkhart, Daniela Brauckhoff, Martin May, and Elisa Boschi. The risk-utility tradeoff for ip address truncation. In *Proceedings of the 1st ACM workshop on Network data anonymization*, NDA '08, pages 23–30, New York, NY, USA, 2008. ACM.

[Chakravarty *et al.*, 2008] Sambuddho Chakravarty, Angelos Stavrou, and Angelos D. Keromytis. Identifying proxy nodes in a tor anonymization circuit. In *Proceedings of the 2008 IEEE International Conference on Signal Image Technology and Internet Based Systems*, SITIS '08, pages 633–639, Washington, DC, USA, 2008. IEEE Computer Society.

[Chaum, 1981] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24:84–90, February 1981.

[Coull *et al.*, 2007a] S. E. Coull, M. P. Collins, C. V. Wright, F. Monrose, and M. K. Reiter. On web browsing privacy in anonymized netflows. In *USENIX Security*, 2007.

[Coull *et al.*, 2007b] Scott Coull, Charles Wright, Fabian Monrose, Michael Collins, and Michael Reiter. Playing devil's advocate: Inferring sensitive information from anonymized network traces. In *Network and Distributed Systems Security Symposium*, 2007.

[Coull *et al.*, 2008] S. Coull, C. Wright, A. Keromytis, F. Monrose, and M. Reiter. Taming the devil: Techniques for evaluating anonymized network data. In *Network and Distributed Systems Security Symposium*, 2008.

[Coull *et al.*, 2009] Scott E. Coull, Fabian Monrose, Michael K. Reiter, and Michael Bailey. The challenges of effectively anonymizing network data. *Conference For Homeland Security, Cybersecurity Applications & Technology*, 0:230–236, 2009.

[Coull *et al.*, 2011] Scott Coull, Fabian Monrose, and Michael Bailey. On Measuring the Similarity of Network Hosts: Pitfalls, New Metrics, and Empirical Analyses. In *Proceed-*

ings of the 18th Annual Network & Distributed System Security Symposium (NDSS '11), San Diego, California, USA, February 2011.

[Cretu *et al.*, 2008] Gabriela F. Cretu, Angelos Stavrou, Michael E. Locasto, Salvatore J. Stolfo, and Angelos D. Keromytis. Casting out demons: Sanitizing training data for anomaly detection. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, May 2008.

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

[Dewaele *et al.*, 2010] Guillaume Dewaele, Yosuke Himura, Pierre Borgnat, Kensuke Fukuda, Patrice Abry, Olivier Michel, Romain Fontugne, Kenjiro Cho, and Hiroshi Esaki. Unsupervised host behavior classification from connection patterns. *Int. J. Netw. Manag.*, 20:317–337, September 2010.

[Dingledine *et al.*, 2004] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: the second-generation onion router. In *Proceedings of the 13th conference on USENIX Security Symposium - Volume 13*, SSYM'04, pages 21–21, Berkeley, CA, USA, 2004. USENIX Association.

[Dwork *et al.*, 2010] Cynthia Dwork, Guy N. Rothblum, and Salil Vadhan. Boosting and differential privacy. *Foundations of Computer Science, Annual IEEE Symposium on*, 0:51–60, 2010.

[Dwork, 2006] Cynthia Dwork. Differential privacy. In *in ICALP*, pages 1–12. Springer, 2006.

[Edman and Yener, 2009] Matthew Edman and Bülent Yener. On anonymity in an electronic society: A survey of anonymous communication systems. *ACM Comput. Surv.*, 42:5:1–5:35, December 2009.

[Fan *et al.*, 2004] Jinliang Fan, Jun xu, Mostafa H. Ammar, and Sue B. Moon. 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.

[Fogla and Lee, 2006] Prahlad Fogla and Wenke Lee. Evading network anomaly detection systems: Formal reasoning and practical techniques. In *Proceedings of the* 13^{th} *ACM Conference on Computer and Communications Security (CCS)*, pages 59–68, 2006.

[Fogla *et al.*, 2006] Prahlad Fogla, Monirul Sharif, Roberto Perdisci, Oleg Kolesnikov, and Wenke Lee. Polymorphic Blending Attacks. In *Proceedings of the USENIX Security Conference*, 2006.

[Force, 1989] Internet Engineering Task Force. Rfc 1122: Requirements for internet hosts – communication layers, October 1989.

[Foukarakis *et al.*, 2009] Michael Foukarakis, Demetres Antoniades, and Michalis Polychronakis. Deep packet anonymization. In *Proceedings of the Second European Workshop on System Security*, EUROSEC '09, pages 16–21, New York, NY, USA, 2009. ACM.

[Freedman and Morris, 2002] Michael J. Freedman and Robert Morris. Tarzan: a peer-to-peer anonymizing network layer. In *Proceedings of the 9th ACM conference on Computer and communications security*, CCS '02, pages 193–206, New York, NY, USA, 2002. ACM.

[Gamer *et al.*, 2008] Thomas Gamer, Christoph P. Mayer, and Marcus Schöller. Pktanon - a generic framework for profile-based traffic anonymization. *PIK Praxis der Informationsverarbeitung und Kommunikation*, 2:67–81, June 2008.

[Ghahramani and Hinton, 1996] Zoubin Ghahramani and Geoffrey Hinton. Parameter estimation for linear dynamical systems. Technical report, University of Toronto, 1996.

[Gönen and Alpaydin, 2011] Mehmet Gönen and Ethem Alpaydin. Multiple kernel learning algorithms. *Journal of Machine Learning Research*, 12:2211–2268, July 2011.

[Hopper and Vasserman, 2006] Nicholas Hopper and Eugene Y. Vasserman. On the effectiveness of k-anonymity against traffic analysis and surveillance. In *Proceedings of the 5th ACM workshop on Privacy in electronic society*, WPES '06, pages 9–18, New York, NY, USA, 2006. ACM.

[Huang *et al.*, 2011] Xin Huang, Fabian Monrose, and Michael K. Reiter. Amplifying limited expert input to sanitize large network traces. In *In Proceedings of IEEE/IFIP International Conference on Dependable Systems and Networks (DSN); Performance and Dependability Symposium (PDS)*. IEEE, June 2011.

[Iliofotou *et al.*, 2007] Marios Iliofotou, Prashanth Pappu, Michalis Faloutsos, Michael Mitzenmacher, Sumeet Singh, and George Varghese. Network monitoring using traffic dispersion graphs (tdgs). In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, IMC '07, pages 315–320, New York, NY, USA, 2007. ACM.

[Jebara *et al.*, 2004] Tony Jebara, Risi Kondor, and Andrew Howard. Probability product kernels. *Journal of Machine Learning Research*, 5:819–844, 2004.

[Jebara *et al.*, 2007] Tony Jebara, Yingbo Song, and Kapil Thadani. Spectral clustering and embedding with hidden markov models. In Joost Kok, Jacek Koronacki, Raomon Mantaras, Stan Matwin, Dunja Mladenic, and Andrzej Skowron, editors, *Machine Learning: ECML 2007*, volume 4701 of *Lecture Notes in Computer Science*, pages 164–175. Springer Berlin / Heidelberg, 2007.

[Kannan *et al.*, 2000] Ravi Kannan, Santosh Vempala, and Adrian Vetta. On clusterings: Good, bad, and spectral. In *41st Annual Symposium on the Foundation of Computer Science*, pages 367–380. IEEE Computer Society, nov "2000".

[Karr *et al.*, 2006] A. F. Karr, C. N. Kohnen, A. Oganian, J. P. Reiter, and A. P. Sanil. A framework for evaluating the utility of data altered to protect confidentiality. volume 60, 2006.

[Kelly *et al.*, 2008] Douglas J. Kelly, Richard A. Raines, Michael R. Grimaila, Rusty O. Baldwin, and Barry E. Mullins. A survey of state-of-the-art in anonymity metrics. In *Proceedings of the 1st ACM workshop on Network data anonymization*, NDA '08, pages 31–40, New York, NY, USA, 2008. ACM.

[Kifer and Machanavajjhala, 2011] Daniel Kifer and Ashwin Machanavajjhala. No free lunch in data privacy. In *Proceedings of the 2011 international conference on Management of data*, SIGMOD '11, pages 193–204, New York, NY, USA, 2011. ACM.

[Kohno *et al.*, 2005] Tadayoshi Kohno, Andre Broido, and K.C. Claffy. Remote physical device fingerprinting. *IEEE Transactions on Dependable and Secure Computing*, 2:93–108, 2005.

[Kolesnikov and Lee, 2006] O. Kolesnikov and Wenke Lee. Advanced polymorphic worms: Evading ids by blending in with normal traffic. In *Proceedings of the USENIX Security Symposium*, 2006.

[Koukis *et al.*, 2006] D. Koukis, S. Antonatos, and K. Anagnostakis. On the privacy risks of publishing anonymized ip network traces. In Herbert Leitold and Evangelos Markatos, editors, *Communications and Multimedia Security*, volume 4237 of *Lecture Notes in Computer Science*, pages 22–32. Springer Berlin / Heidelberg, 2006. 10.1007/119090333.

[Kruskal and Wish, 1978] J.B. Kruskal and M. Wish. *Multidimensional Scaling*. Sage, 1978.

[Lakkaraju and Slagell, 2008] Kiran Lakkaraju and Adam Slagell. Evaluating the utility of anonymized network traces for intrusion detection. In *Proceedings of the* 4^{th} *international conference on Security and privacy in communication netowrks*, 2008.

[McSherry and Mahajan, 2010] Frank McSherry and Ratul Mahajan. Differentially-private network trace analysis. *SIGCOMM Comput. Commun. Rev.*, 40:123–134, August 2010.

[Meila, 2007] Marina Meila. Comparing clusterings – an information based distance. In *Journal of Multivariate Analysis*, 2007.

[Minshall, 2005] Greg Minshall. TCPdpriv. http://ita.ee.lbl.gov/html/contrib/tcpdpriv.html, October 2005.

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

[Mogul and Arlitt, 2006] Jeffrey C. Mogul and Martin Arlitt. Sc2d: an alternative to trace anonymization. In *Proceedings of the 2006 SIGCOMM workshop on Mining network data*, 2006.

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

[Ng *et al.*, 2001] Andrew Ng, Michael Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems*, 2001.

[Ostermann, 2003] Shawn Ostermann. Tcptrace, 2003.

[Pang *et al.*, 2005] Ruoming Pang, Mark Allman, Michael Bennett, Jason Lee, and Vern Paxson. A first look at modern enterprise traffic. In *Proceedings of the* 5^{th} *ACM SIGCOMM conference on Internet Measurement*, 2005.

[Pang *et al.*, 2006] Ruoming Pang, Mark Allman, Vern Paxson, and Jason Lee. The devil and packet trace anonymization. *SIGCOMM Comput. Commun. Rev.*, 36:29–38, January 2006.

[Parate and Miklau, 2008] Abhinav Parate and Gerome Miklau. A framework for utility-driven network trace anonymization. Technical report, University of Massachusetts, Amherst, 2008.

[Parate and Miklau, 2009] Abhinav Parate and Gerome Miklau. A framework for safely publishing communication traces. In *Proceeding of the* 18^{th} *ACM conference on Information and knowledge management*, 2009.

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

[Rabiner, 1989] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, pages 257–286, 1989.

[Rosenberg and Hirschberg, 2007] Andrew Rosenberg and Julia Hirschberg. V-measure: A conditional entropy-based external cluster evaluation measure. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2007.

[Roweis and Saul", 2000] "S. Roweis and L. Saul". Nonlinear dimensionality reduction by locally linear embedding, "2000".

[Shaw and Jebara, 2007] Blake Shaw and Tony Jebara. Minimum Volume Embedding. In *AISTATS*, pages 460–467, March 2007.

[Shi and Malik, 2000] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:888–905, 2000.

[Song *et al.*, 2009] Yingbo Song, Angelos D. Keromytis, and Salvatore J. Stolfo. Spectrogram: A Mixture-of-Markov-Chains Model for Anomaly Detection in Web Traffic. In *Proceedings of the* 16^{th} *Symposium on Network and Distributed System Security (NDSS)*, February 2009.

[Song *et al.*, 2011] Yingbo Song, Salvatore J. Stolfo, and Tony Jebara. Behavior-based network traffic synthesis. In *IEEE Conference on Technologies for Homeland Security.*, November 2011.

[Spirent, 2011] Spirent. Spirent hypermetrics 40/100g module, 2011.

[Syverson *et al.*, 2001] Paul Syverson, Gene Tsudik, Michael Reed, and Carl Landwehr. Towards an analysis of onion routing security. In *International workshop on designing privacy enhancing technologies: design issues in anonymity and unobservability*, pages 96–114. Springer-Verlag New York, Inc., 2001.

[Tan *et al.*, 2003] Godfrey Tan, Massimiliano Poletto, John Guttag, and Frans Kaashoek. Role classification of hosts within enterprise networks based on connection patterns. In *In USENIX Annual Technical Conference*, pages 15–28, 2003.

[tcp, 2011] Tcpdump: a powerful command-line packet analyzer; and libpcap, a portable c/c++ library for network traffic capture., 2011. http://www.tcpdump.org.

[Tony Jebara, 2007] Kapil Thadani Tony Jebara, Yingbo Song. Density estimation under independent similarly distributed sampling assumptions. In *Neural Information Processing Systems (NIPS)*, 2007.

[Tor, 2011] Tor project: Anonymity online. http://www.torproject.org/, 2011.

[Troncoso and Danezis, 2009] Carmela Troncoso and George Danezis. The bayesian traffic analysis of mix networks. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pages 369–379, New York, NY, USA, 2009. ACM.

[Truta *et al.*, 2004] Traian Marius Truta, Farshad Fotouhi, and Daniel Barth-Jones. Assessing global disclosure risk in masked microdata. In *Proceedings of the 2004 ACM workshop on Privacy in the electronic society*, WPES '04, pages 85–93, New York, NY, USA, 2004. ACM.

[United States Military Academy, 2009] United States Military Academy. ITOC CDX Research Dataset, 2009.

[Vishwanathan *et al.*, 2010] S. V. N. Vishwanathan, Nicol N. Schraudolph, Risi Kondor, and Karsten M. Borgwardt. Graph kernels. *Journal of Machine Learning Research*, 11:1201–1242, August 2010.

[Wang *et al.*, 2005] Ke Wang, Gabriela Cretu, and Salvatore J. Stolfo. Anomalous Payload-based Worm Detection and Signature Generation. In *Proceedings of the* 8^{th} *International Symposium on Recent Advances in Intrusion Detection (RAID)*, pages 227–246, September 2005.

[Wang *et al.*, 2006] Ke Wang, Janak J. Parekh, and Salvatore J. Stolfo. Anagram: A Content Anomaly Detector Resistant To Mimicry Attack. In *Proceedings of the* 9^{th} *International Symposium on Recent Advances in Intrusion Detection (RAID)*, pages 226–248, 2006.

[Weinberger and Saul, 2006] Kilian Q. Weinberger and Lawrence K. Saul. Unsupervised learning of image manifolds by semidefinite programming. *Int. J. Comput. Vision*, 70:77–90, October 2006.

[Wright *et al.*, 2006] Charles V. Wright, Fabian Monrose, and Gerald M. Masson. On inferring application protocol behaviors in encrypted network traffic. *Journal of Machine Learning Research*, 2006.

[Wright *et al.*, 2009] Charles V. Wright, Scott E. Coull, and Fabian Monrose. Traffic morphing: An efficient defense against statistical traffic analysis. In *Proceedings of the Network and Distributed Security Symposium - NDSS '09*. IEEE, IEEE, February 2009.

[Xiao *et al.*, 2010] Xiaokui Xiao, Yufei Tao, and Nick Koudas. Transparent anonymization: Thwarting adversaries who know the algorithm. *ACM Trans. Database Syst.*, 35:8:1–8:48, May 2010.

[Yin and Yang, 2005] Jie Yin and Qiang Yang. Integrating hidden markov models and spectral analysis for sensory time series clustering. In *Proceedings of the Fifth IEEE International Conference on Data Mining*, ICDM '05, pages 506–513, Washington, DC, USA, 2005. IEEE Computer Society.

[Yurcik *et al.*, 2007] William Yurcik, Clay Woolam, Greg Hellings, Latifur Khan, and Bhavani Thuraisingham. Scrub-tcpdump: A multi-level packet anonymizer demonstrating privacy/analysis tradeoffs. In *Third International Conference on Security and Privacy in Communications Networks (SecureComm)*, 2007.

[Zalewski, 2006] Michal Zalewski. p0f: Passive OS fingerprinting tool, 2006.

[Zhang *et al.*, 2007] Lei Zhang, Sushil Jajodia, and Alexander Brodsky. Information disclosure under realistic assumptions: privacy versus optimality. In *Proceedings of the 14th ACM conference on Computer and communications security*, CCS '07, pages 573–583, New York, NY, USA, 2007. ACM.

[Zhang *et al.*, 2008] Kai Zhang, Ivor W. Tsang, and James T. Kwok. Improved nyström low-rank approximation and error analysis. In *Proceedings of the 25th international conference on Machine learning*, ICML '08, pages 1232–1239, New York, NY, USA, 2008. ACM.