An Empirical Evaluation of Entropy-based Anomaly Detection

George Nychis

May, 2007

Information Networking Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Hui Zhang
David G. Andersen

Submitted in partial fulfillment of the requirements
for the degree of Master of Science

Keywords: anomaly detection, entropy, traffic features, wavelet, signal analysis, correlation

Abstract

There is considerable interest in using entropy-based analysis of traffic feature distributions for anomaly detection. Entropy-based metrics are appealing since they provide more fine-grained insights into traffic structure than traditional traffic volume analysis. While previous work has demonstrated the benefits of using the entropy of different traffic distributions in isolation to detect anomalies, there has been little effort in comprehensively understanding the detection power provided by entropy-based analysis of multiple traffic distribution used in conjunction with each other.

We compare and contrast the anomaly detection capabilities provided by different entropy-based metrics. We consider two classes of distributions: flow-header features (IP addresses, ports, and flow-sizes), and behavioral features (out- and in-degree of hosts measuring the number of distinct destination/source IP addresses that each host communicates with). Somewhat surprisingly, we observe that the entropy of the address and port distributions are strongly correlated with each other, and also detect very similar anomalies in our traffic trace. The behavioral and flow size distributions appear less correlated and detect incidents that do not show up as anomalies among the port and address distributions. Further analysis using synthetically generated anomalies also suggests that the port and address distributions have limited utility in detecting scan and bandwidth flood anomalies. Based on our results we derive implications for selecting traffic distributions in entropy-based anomaly detection.

In support of the thesis and future work, we present the Datapository Anomaly Detection Testbed, a framework and storage facility for analyzing and developing detection methods, generating and labeling anomalies, and analyzing traffic features with user provided traffic sets or publicly available traffic sets in the Datapository database. Through the collaboration of future users, we hope to expand the set of available detection methods, synthetic anomaly models, and publicly available traffic data and tools for analysis.

To the Greeks, whose support and dancing gets me through the day.

Acknowledgments

After a year of digging aimlessly at network traffic, I found something! However, finding that something could not have been possible without the encouragement, time, resources, and effort many people provided me along the way.

First, I thank my advisor, Hui Zhang, whose teaching and research has been inspirational to me. The first advice Hui gave me was to ”not get lost in the data.” Those words took on a whole new meaning after Hui gave me access to terabytes of flow level traffic and I found myself wandering down paths completely unrelated to anomaly detection. However, Hui’s advice has been golden to me and has been able to keep me on track when I have wandered astray. I am very grateful for the opportunity to work with Hui and his continuing support of me.

I don’t remember whether it was what I learned from David Andersen in 15-441: Computer Networks, his research, or when he literally jumped off a wall outside his office while talking to me that I decided to ask him to also advise my work. I suspect it was one of the prior, however the latter is an certainly an unforgettable memory. Dave has sincerely been influential and motivational to me, and the level of enthusiasm he puts in to everything he does is truly unique and inspirational. The guidance and ideas he has given me throughout the year on this thesis are irreplaceable. Thank you.

From the first day Hui introduced me to Vyas Sekar, I have been filled with advice, ideas, and support which have been invaluable to not only this thesis, but my graduate career. Vyas has patiently introduced me to the many, and often overwhelming, aspects of research that I continue to share with those around me. Thank you for your patience, the writing lessons, and the many things you have taught me.

When I said this work could not have been possible without the time and resources many people have provided me, I did not specify either were related directly to this thesis. In fact, the work would not have been possible without my time away from it. Many thanks goes to JJ Stamatelos and Shawn Robinson who ensured I got my weekly (unfortunately not daily) dosage of sunlight. I am extremely grateful for the many occasions both of you forceful removed me from the lab, dropped me off food to eat, and provided me advice on that thing that always seems to get in the way of the work I enjoy, life. The resources that made this work possible are the cookies Korina Anna Loumidi freshly baked for me multiple times a week. Although the first batch came out horribly burnt, my faith in her paid off and I received hundreds of flawless cookies throughout the past year. Similar thanks goes to Alex, Stacey, Evan, Tharina, George, Joey, Tom, Maria, Hara, Mike, Brock, Melissa, Dan, Thibaud, Aroon, Doug, and Nick.

My final and most sincere thanks goes to my parents, Peter and Vicki Nychis. This work

literally could not have been possible without their support. Although I am only 15 miles from home, I am sure we would all agree it has felt like hundreds since I have become a graduate student. The many visits to Oakland, home cooked meals, advice, and support never goes unappreciated. I love you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Researchers have recently proposed the use of entropy based metrics for traffic analysis [36] and anomaly detection [14, 4, 9, 6, 16, 34]. Entropy based anomaly detection metrics capture more fine-grained traffic patterns than simple traffic volume based metrics. Previous measurement studies have shown that entropy based anomaly detection can detect many anomalies that do not manifest themselves as large deviations in aggregate traffic volume [14].

Many traffic features (e.g., flow size, ports, addresses) have been suggested as candidates for entropy based anomaly detection. However, there has been little work in understanding the detection capabilities provided by a set of entropy metrics used in conjunction with one another. Specifically, it is not clear if the different entropy based metrics proposed so far complement each other in their detection capabilities, or if they provide redundant functionality.

The goal of this paper is to provide a better understanding about the use of entropy-based traffic anomaly detection using different traffic features. We consider two classes of traffic feature distributions: *flow-header* features, and *behavioral* features. The flow-header features are addresses (both source and destination), ports (both source and destination) and the flow size distribution (FSD). These have been suggested as good candidates for detecting worms, scans, and DDoS attacks [14, 6, 13]. The behavioral features are the in and out-degree distributions, where the degree of an end-host X is the number of distinct IP addresses that X communicates with. The behavioral distributions capture the structure of the end-host communication patterns.

The primary dataset used in this evaluation is a month-long traffic trace collected within a large university network having in excess of 60,000 active IP addresses. The trace consists of more than 2.5 billion flows comprising a total traffic volume greater than 90 TB. Our key analysis and measurement results are:

- The port and address distributions are very correlated, with pairwise correlation scores greater than 0.95. The degree distributions and FSD exhibit weak correlations with each other and with the port/address distributions.

- The anomalies detected by the port and address distributions overlap significantly. In our dataset, almost all the anomalies detected by these distributions are *alpha flows* [14]. In contrast, the host degree distributions and FSD identify anomalies such as abnormal scanning, DoS, and peer-to-peer activity that are not detected by the port and address distributions.

- We complement the measurement study by evaluating several synthetic anomaly scenarios using the collected flow data as background traffic. FSD and the degree distributions detect scanning events whereas the port and address distributions do not. For bandwidth flood and DDoS events, the port and degree distributions detect only high-magnitude events that would appear as traffic volume anomalies anyway.

While ports and addresses have been commonly suggested [14] as good candidates for entropy-based anomaly detection, our results give us reason to question this rationale. Our results also suggest a natural metric for choosing traffic features for entropy based anomaly detection: select traffic distributions that are inherently complementary to one another and thus provide different views into the underlying traffic structure. For example, we find that the behavioral distributions and the FSD, which are qualitatively different from the port and address distributions, provide distinct and often better anomaly detection capabilities. These complementary distributions can detect anomalies embedded in one another. As a specific example, we found one instance of two anomalies occurring concurrently – one alpha flow detected by the port and address distributions, and an unrelated peer-to-peer anomaly detected by FSD alone. Our results thus suggest that the selection of traffic distributions in entropy-based anomaly detection should be made more judiciously, and in particular we should look beyond simple port and address based distributions.

The rest of the paper is organized as follows. Section 2 sets up the preliminary definitions and anomaly detection methods we use in our analysis. Section 3 presents our measurement results. We also augment our measurement study using synthetic anomalies in Section 3.6. We conclude in Section 6 discussing the implications of our measurement results after reviewing related work in Section 5.

Chapter 2

Preliminaries

First, we define the notion of normalized entropy used for anomaly detection. Next, we briefly describe the timeseries analysis techniques for anomaly detection that we borrow from existing work. We also introduce the different traffic distributions we evaluate.

2.1 Entropy

Let X denote a random variable representing the distribution of values a particular traffic feature (e.g., the source address or destination port of a flow) can take. Let $x_1 \ldots x_N$ denote the range of values that X can take, and for each x_i let $p(x_i)$ represent the probability that the random variable X takes the value x_i, i.e., $p(x_i) = Pr[X = x_i]$. The entropy [30] of the random variable X is defined as[1]:

$$H(X) = -\sum_{i=1}^{N} p(x_i) \log p(x_i) \tag{2.1}$$

Normalized Entropy: Since some items may not appear during a single measurement interval we define N_0 to be the number of distinct items that are actually present in the given measurement interval. Intuitively, the entropy is a measure of the diversity of the data coming over the stream. The entropy attains its minimum value of zero when all the items coming over the stream are the same and its maximum value of $\log(N_0)$ when each item in the stream appears exactly once. Across measurement intervals we might observe a different number of distinct items (N_0). Thus, we normalize H to be between zero and one by computing the normalized entropy: $H/\log N_0$. This normalization measures the relative randomness within each measurement interval, and allows us to quantitatively compare entropy values across time. For the remainder of the discussion we will use this definition of normalized entropy.

[1] All logarithms in this paper are to the base 2 and we define $0 \log 0 = 0$

2.2 Anomaly Detection Methods

We use two independent methods for timeseries anomaly detection. We do so primarily to avoid any biases arising from a particular anomaly detection technique. Our goal is not to evaluate the timeseries analysis methods themselves; rather, our goal is to demonstrate that our observations regarding the properties of entropy-based anomaly detection are independent of the underlying anomaly detection technique used.

Wavelet analysis: Barford et al. propose wavelet analysis for detecting anomalies in traffic timeseries data [3]. Their technique treats measurement data (e.g., number of packets per five minute interval) as a generic timeseries signal. Using wavelet decomposition, the timeseries is then decomposed into different sub-components consisting of the low, mid-range, and high-frequency components. The insight behind wavelet analysis is that the low frequency components capture expected traffic trends (e.g., the mean and the diurnal traffic patterns), while the mid to high frequency components capture instantaneous variations in the timeseries (i.e., deviations and anomalies from the expected patterns).

After this decomposition, the mid-range and high frequency components are normalized to have unit variance. Next, they compute the local variability of the high frequency and mid frequency components using sliding windows of different sizes. The size of the sliding window determines the duration of anomalies that can be detected. The local variability of the high frequency and mid frequency parts are then added together. The final anomaly detection step is a simple thresholding function applied to this combined signal. A sliding window of size 6 is used which corresponds to a 30 minute period given that each data point represents a five minute interval.

Heuristic Threshold-based detection: As an alternative to wavelet based detection, we consider a heuristic technique adapted from prior work [26, 29]. The high-level goal is to estimate the mean and standard deviation of the timeseries signal using historical data. For each future observation, we compute a deviation score: $Score = \frac{|Observation - Mean|}{Stddev}$. This score captures how far away from the mean value a particular observation is, expressed relative to the standard deviation. We flag an anomaly whenever any observation has a score greater than some threshold α.

If the mean and standard deviation are calculated with historical traffic data that contain large anomalies, then these values are likely to over-estimate the true statistics. Therefore, this heuristic may miss anomalies in future observations, because the model of traffic accommodates more anomalies than it should. To avoid this bias, we introduce an iterative cleaning technique for learning the mean and standard deviation given possibly noisy training data. The approach works as follows. In each iteration, we compute the mean and the standard deviation. For the current iteration, we find anomalous datapoints, i.e., those that are greater than $\alpha = 3$ standard deviations away from the mean. We remove these anomalies from consideration for further iterations. The iteration continues until the mean and standard deviation obtained are *stable*, meaning that the values do not differ significantly across subsequent iterations. In our experiments, we find that this process usually terminates in 4-5 iterations and drops fewer than 3% of the data points.

StartTime(sec)	EndTime(sec)	Proto	Left_IP:Port	Flow_Dir	Right_IP:Port	Src_Pkts	Dst_Pkts	Src_Bytes	Dst_Bytes
12802	12803	UDP	244.0.0.1:3180	->	128.2.240.216:139	6	2	372	120
12820	12824	TCP	0.2.122.42:9478	->	109.173.146.198:22	29	30	1566	3772
12824	12824	TCP	130.20.143.124:80	<-	0.2.122.42:2118	2	1	846	60
12824	12825	TCP	0.2.122.42:2341	->	109.173.146.198:22	31	40	1662	4201
12825	12825	UDP	244.0.0.1:2718	->	0.2.122.42:139	6	3	372	150

Figure 2.1: Flow level data format used for the analysis including sample records.

2.3 Traffic Features For Anomaly Detection

In our study we compare seven traffic distributions. Five of these are obtained from flow-header features: source address, destination address, source port, destination port, and the flow size distribution (FSD). The remaining two are based on structural properties of inter-host communication behavior: the in-degree and out-degree distribution across hosts, where the degree of an end-host X is defined as the number of distinct IP addresses that X communicates with.

Prior work on using flow-header features in traffic analysis uses unidirectional flow information commonly exported by most routers [21, 23]. Our dataset contains bidirectional flow records [2]; we explicitly convert each bidirectional flow record into two unidirectional flows. Consider Figure 2.1 as an example. The first flow entry would be interpreted as two flows, one flow was from 244.0.0.1 to 128.2.240.216 with a total of 6 packets, the other flow was from 128.2.240.216 to 244.0.0.1 with a total of 2 packets.

The behavioral metrics use *directional* traffic flows to distinguish between incoming and outgoing connections for a single host.

Feature examples using Figure 2.1:

- *Address* 0.2.122.42 sourced 65 packets and was the destination of 77 packets
- *Port* 22 sourced 70 packets and was the destination of 60
- The *traffic volume* of the example was 74 source packets and 76 destination packets
- 3 hosts had an *in degree* of 1, 2 hosts had an *out degree* of 1
- 2 flows had a *flow size* of 6 packets

Computing normalized entropy:

- *Addresses:* For each source (destination) IP address x_i, we calculate the probability

$$p(x_i) = \frac{\text{Number of pkts with } x_i \text{ as src (dst) address}}{\text{Total number of pkts}}$$

  The normalization factor is $\log(N)$, where N is the number of active source (destination) addresses observed during the measurement interval.

- *Ports:* For each source (destination) port x_i, we calculate the probability:

$$p(x_i) = \frac{\text{Number of pkts with } x_i \text{ as src (dst) port}}{\text{Total number of pkts}}$$

  The normalization factor is $\log(P)$, where P is the total number of distinct active source (destination) ports for the interval.

- *Flow size distribution:* For each actual value that flow size (measured in packets) takes, we calculate the probability:

$$p(x_i) = \frac{\text{Number of flows with flowsize } x_i}{\text{Total number of flows}}$$

  The normalization factor is $\log(F)$, where F is the number of distinct flow sizes we observe within the measurement interval.

- *Behavioral distributions:* For a host X, the *out-degree* is the number of distinct IP addresses that X contacts, and the *in-degree* is the number of distinct IP addresses that contact X. Each X is an active internal IP address inside the network under consideration (e.g., in our dataset we only consider hosts inside the university): these are the only hosts for which we have a complete view of both incoming and outgoing traffic. For each value of out-degree (in-degree) x_i, we calculate the probability

$$p(x_i) = \frac{\text{Number of hosts with out-degree } x_i}{\text{Total number of hosts}}$$

  The normalization factor for the out-degree distribution is $\log(D)$, where D is the number of distinct out-degree (similarly in-degree) values observed during the measurement interval.

Chapter 3

Measurement Results

Dataset: Our analysis is based on Argus [2] flow level data captured at a core router within Carnegie Mellon University[1]. The data was captured in February 2005. The data set contains traffic to and from over 100,000 active IP addresses consisting of a total aggregate traffic of 92 TB over approximately 2.5 billion flows. IP addresses in the dataset were anonymized [35]. Because the anonymization preserves a one-to-one mapping between the unanonymized and anonymized IP address for the entire trace, the anonymization does not affect our analysis (traffic feature distributions remain identical after anonymization). Application ports were not anonymized.

The dataset is split into five minute non-overlapping intervals. Each bin consists of flows that completed within the interval. Each flow record includes the connection time, the protocol used, the connection state, flow direction, and source/destination pairs for all of the following: IP address, port, packet count, and byte count. The format of the data with some example flow records is shown in Figure 2.1. As discussed earlier, to obtain the in- and out-degree distributions, the flow record must indicate the flow's direction. However, in some cases the directionality is not evident from the flow record (e.g., UDP flows, long-lived TCP flows that extend beyond the flow timeout). In such cases, we use application port numbers to infer flow direction[2]. If the directionality is still ambiguous, we arbitrarily select the left host in the flow record to be the originator of the flow.

Roadmap: Our measurement results are structured as follows:

- To understand if different traffic feature distributions are structurally correlated, we compute pair-wise correlation coefficients for the different entropy timeseries values in Section 3.1.

- We analyze if the correlations from Section 3.1 also translate into the space of anomalies by computing pair-wise correlation coefficients for the anomaly deviation scores in Section 3.2.

[1]The router observes all traffic between university hosts and external Internet hosts. Additionally, it also routes a significant fraction of internal inter-departmental traffic

[2]The rationale behind using the port numbers for determining the directionality of a flow is the following. If a host is running a well-known application service, then it is likely to be the server-host in the connection. Since the client that initiates a connection to the server in the majority of client-server transactions we assume that the host that does not use the well-known port is the originator of the connection.

- We analyze the overlap between the set of anomalies detected by the different traffic features in Section 3.3 to understand if the different entropy metrics provide similar detection capabilities or if they differ significantly.

- We use a heuristic approach for identifying the traffic flows contributing to the anomalies in Section 3.4. By analyzing the anomalous flows, we explain why some of the anomalies are detected by several entropy metrics, while other anomalies are unique to specific metrics. We discover several interesting anomalies, including possible botnet activity, the arrival of a P2P "supernode", and a possible outbound spoofed DoS attack.

3.1 Correlations in Entropy Timeseries

Table 3.1 shows the pairwise correlation scores between the entropies of different distributions. We find strong correlations (> 0.95) between the address and port distributions. The remaining metrics show moderate (e.g., FSD and in-degree) to no correlation. Figure 3.1 shows the entropy timeseries values over the entire month-long trace. The visual confirmation of the correlations is just as striking as the values themselves. The degree-based entropies show more short-term stochastic variation than the traffic feature distributions and as such the entropy of the host behavioral distributions do not appear strongly correlated with the other features. Additionally, we observe that many of the spikes and deviations in the timeseries plots are also highly correlated. We will revisit these anomalies in the subsequent discussions.

	OutDeg	SrcAddr	DstAddr	SrcPort	DstPort	FSD
InDeg	0.102	0.100	0.097	0.000	0.007	0.414
OutDeg	-	-0.034	-0.033	-0.054	-0.015	-0.018
SrcAddr	-	-	0.994	0.962	0.956	0.307
DstAddr	-	-	-	0.966	0.969	0.286
SrcPort	-	-	-	-	0.989	0.171
DstPort	-	-	-	-	-	0.181

Table 3.1: Correlation of entropy data with no measurement anomalies.

	DstAddr	SrcPort	DstPort	FSD
SrcAddr	0.988	0.793	0.819	0.214
DstAddr	-	0.761	0.834	0.253
SrcPort	-	-	0.848	0.087
DstPort	-	-	-	0.032

Table 3.2: Correlation of entropy values for Internet2 traffic captured at the Houston router.

To confirm that these results are not an artifact of our dataset, we perform similar analysis using data from the Internet2 [7] backbone. We use two weeks (Dec. 1-14, 2006) of flow traces collected from the eleven backbone routers, and compute the entropy of the flow-header distributions. The

correlations for flow data from the Houston router are shown in Table 3.2. We do not compute the degree based metrics because the sampled Netflow traffic is not directional. The correlation scores shown for the single router are qualitatively representative of other routers. This confirms that the strong correlations we observe are not unique to our dataset.

3.2 Correlations in Anomaly Deviation Scores

Next, we explore if the correlations in the entropy timeseries values also extend into the anomaly space. We first assign anomaly deviation scores to each data point using the techniques described in Section 2.2[3].

	OutDeg	SrcAddr	DstAddr	SrcPort	DstPort	FSD
InDeg	0.248	0.199	0.188	0.185	0.156	0.507
OutDeg	-	0.179	0.165	0.143	0.122	0.396
SrcAddr	-	-	0.991	0.971	0.964	0.319
DstAddr	-	-	-	0.970	0.971	0.300
SrcPort	-	-	-	-	0.986	0.256
DstPort	-	-	-	-	-	0.220

Table 3.3: Pairwise correlations in the wavelet deviation scores.

Table 3.3 shows that the port and address distributions are as strongly correlated in terms of the deviation scores as they are in terms of the raw entropy values. Interestingly, the behavioral features become slightly more correlated to the other metrics. For example, the correlation between out-degree and FSD increases by 0.414 (Table 3.4). We hypothesize the reason for this increase is that the in and out-degree distributions show more stochastic variations than the other distributions. Thus, they tend to be uncorrelated in terms of the timeseries values. However, the wavelet anomaly detection technique removes these noisy variations, and they become more correlated in terms of the deviation scores.

	OutDeg	SrcAddr	DstAddr	SrcPort	DstPort	FSD
InDeg	0.146	0.100	0.091	0.185	0.149	0.093
OutDeg	-	0.213	0.198	0.197	0.137	0.414
SrcAddr	-	-	-0.003	0.009	0.008	0.012
DstAddr	-	-	-	0.003	0.002	0.014
SrcPort	-	-	-	-	-0.002	0.085
DstPort	-	-	-	-	-	0.039

Table 3.4: Difference in correlation values between the heuristic anomaly detection approach and wavelet based anomaly detection.

Threshold based detection also produces similar results. We present the difference from correlations of threshold deviation scores to wavelet based deviation scores in Table 3.4. Our goal is not

[3]For wavelet analysis, the score assigned is the magnitude of localized variance computed over a sliding window of size 6.

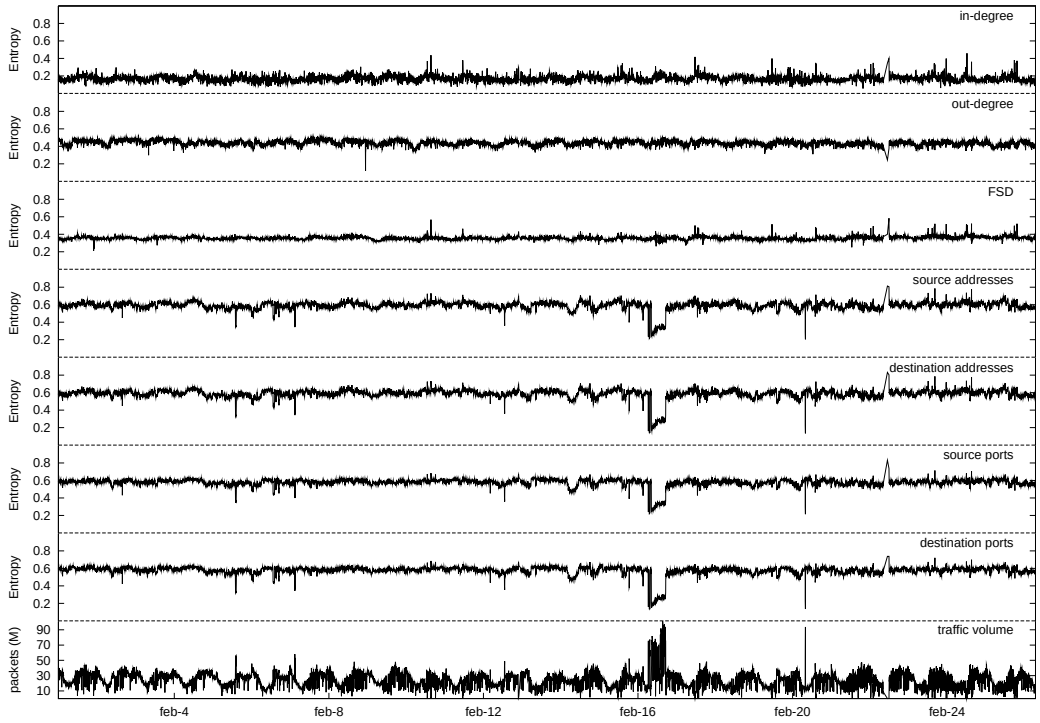

Figure 3.1: Time series of entropy data for February 2005.

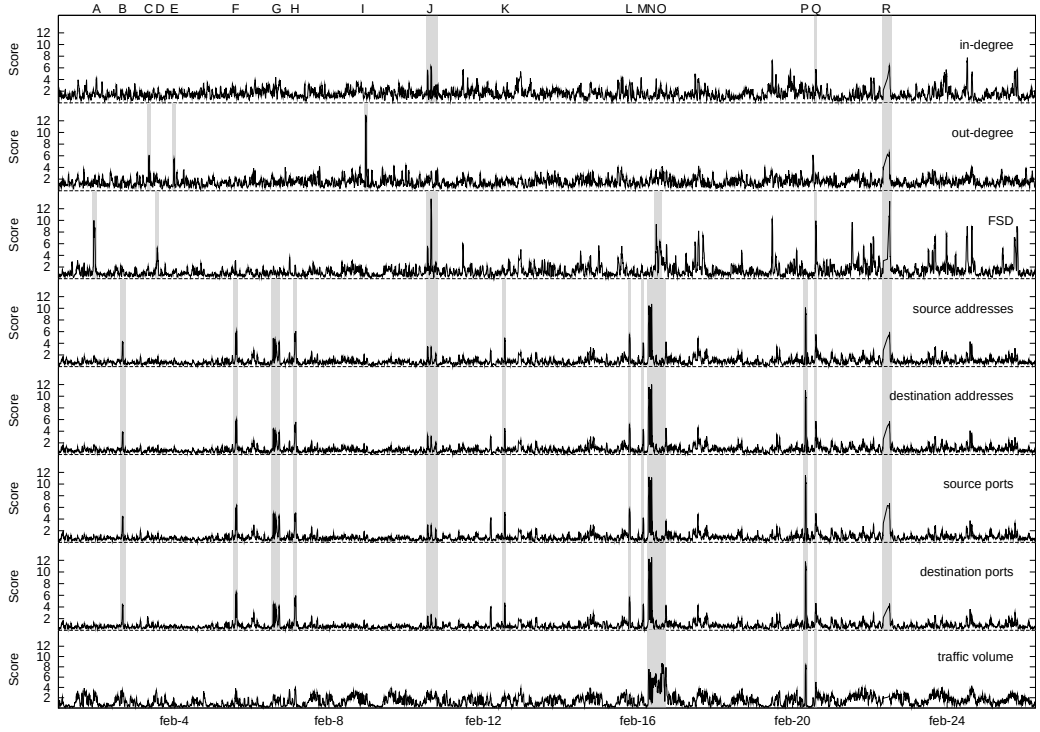

Figure 3.2: Time series of deviation scores computed with the wavelet analysis technique

to show similarities or differences in the anomaly detection methods themselves, but that regardless of the detection method chosen, the patterns of correlation across the entropy based metrics are unaffected. Table 3.4 confirms that there is no bias introduced by choosing one anomaly detection technique over another. The only qualitative difference we observe between the two methods is in the behavioral features. where the significant positive correlation differences from threshold to wavelet detection confirms wavelet analysis' ability to remove noise and diurnal effects. The negligible differences of correlation in deviation scores between the other metrics and methods illustrates that the results are independent of the detection method. Hence, we only present the results from wavelet based detection in the rest of the paper.

3.3 Overlap in Anomalies Detected

Our next goal is to understand the overlap between the detection capabilities provided by different entropy metrics. This helps us quantify the extent to which the different entropy metrics provide similar anomaly detection functionality.

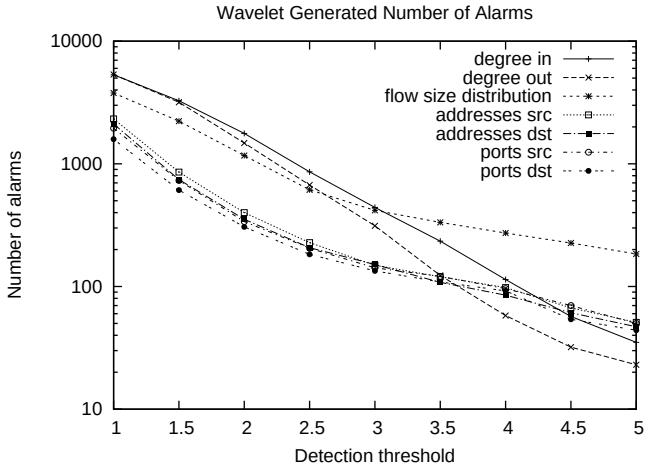

Figure 3.3: The number of alarms generated by wavelet detection at different thresholds

We generate anomalies by specifying a detection threshold α, and flagging any observation that has an anomaly deviation score greater than the threshold α. Figure 3.3 shows the number of anomalies identified for each of the entropy metrics over the entire month as a function of the threshold α. We observe that the behavioral metrics and FSD generate $2\times$ more alarms than the port and address distributions.

After generating the alarms we assign an anomaly overlap score as follows. We construct a timeseries of anomaly incidents for each entropy metric, where a timeslot (five minute bin) is assigned a 1 if there is an anomaly reported and a 0 if there is no anomaly. Thus, we have a vector of 0-1 values representing the entire sequence of anomalies for a entropy metric. We compute the correlations for every pair of (binary) anomaly vectors. By construction, two metrics with the exact same set of anomalies have a correlation score of 1, and metrics that have the exact opposite set of

	OutDeg	SrcAddr	DstAddr	SrcPort	DstPort	FSD
InDeg	0.137	0.197	0.183	0.154	0.126	0.354
OutDeg	-	0.072	0.071	0.026	0.023	0.213
SrcAddr	-	-	0.946	0.864	0.854	0.141
DstAddr	-	-	-	0.858	0.891	0.123
SrcPort	-	-	-	-	0.934	0.087
DstPort	-	-	-	-	-	0.060

Table 3.5: Overlap between anomalies with a threshold $\alpha = 3$.

alarms have a correlation score of -1. We generate these pairwise overlap scores for the anomaly vectors using different values of α.

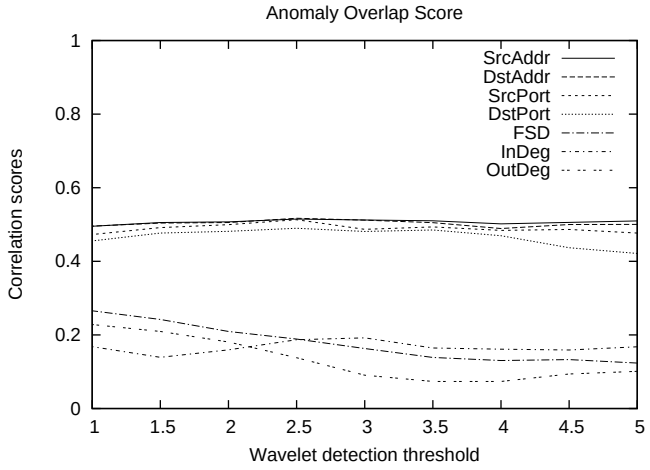

Figure 3.4: Average anomaly overlap score vs. the detection threshold. The lower two lines represent the degree metrics.

Table 3.5 presents the overlap scores for $\alpha = 3$. We notice that the correlation of deviation scores translates to overlap of anomalies in the detection space. Furthermore, the overlap is independent of the value of α. Even at very low deviation scores, where thousands of alarms are generated, the alarms are just as correlated as using larger thresholds. For clarity, we do not present the full correlation matrices for different thresholds but instead report the average for each entropy metric as a function of the threshold α in Figure 3.4. The figure illustrates that the correlations in alarms stay relatively constant across all thresholds.

3.4 Understanding Anomalies In-Depth

Why do the anomalies detected by the port and address distributions overlap and why do FSD and the degree distributions provide unique detection capabilities? To answer these questions we use a heuristic approach to identify for each anomaly event, the set of traffic flows that contribute to the

Anomaly Type	Affected Metrics	Labels
Alpha Flows (Botnet activity)	Addresses, Ports	B, F-H, K-N, P
Scans	FSD	A, D
P2P Supernode Activity	FSD	O
Spoofed DoS	Degree	C, E, I
Measurement Outage	Inconsistent	J, Q, R

Table 3.6: Labeled traffic anomalies.

particular anomaly. Once we identify the actual anomalous flows, we explain the effect these have on different traffic distributions, and why these manifest as anomalies in specific traffic features.

We discuss eighteen events, indicated by alphabetical labels in Figure 3.2, and summarized in Table 3.6. (Some anomalies span multiple timeslots; we cluster anomalies close in time into a single event.) Among these, we find three measurement anomalies that do not exhibit any common characteristics among the distributions. We discover several alpha flows [14] that explain the strong overlap between the anomalies detected by the address and port distribution. The remaining anomalies are unique to FSD and the in-degree distribution. Through our labeling methodology, we also discover an embedded anomaly events which would have gone undetected using volume based detection.

Labeling Method: In the absence of ground truth for our dataset, we develop a semi-automated approach for discovering anomalous traffic flows. While our approach is a heuristic, it helps us explain more than 90% of the observed anomalies.

We analyze the *top-k* contributors within each distribution (e.g., destination address, in-degree) for the timeslots when the anomalies occur. The rationale behind this approach is that during an anomaly, the top few contributors to specific traffic distributions change significantly. If some discrepancy is detected in the top-k value (e.g., if a new value enters the top-k, or if the contribution of the top few values changes significantly), it is treated as a possible cause of the anomaly. The set of traffic flows associated with this discrepancy are then identified and explicitly removed. After the suspected anomalous flows are removed from the trace, the entropy and wavelet scores are recomputed using the remaining traffic flows. If the anomaly subsides, we are confident that these anomalous flows did indeed cause the original anomaly.

To illustrate, consider a large bandwidth flood destined to host X. Since this host receives significantly more traffic than other hosts, X is likely to appear in the top-k of the destination address distribution. The flows having X as the destination host are then removed, and the entropy and wavelet values are recomputed. If the deviation scores for the new entropy values decrease below the detection threshold α, the bandwidth flood to X is considered the cause of the original anomaly.

Measurement Anomalies: Events J, Q, and R are measurement anomalies, characterized by few to no flow records in our dataset. The measurement anomalies do not show any consistent behavior across the different traffic features. This is illustrated in event R where all metrics are able to detect the anomaly using a threshold $\alpha = 4$ except for the out-degree. As measurement anomalies are not interesting from a traffic structure point of view, they are not further discussed.

Alpha-Flows (likely botnet activity): In alpha flows, a small number of ports and addresses (both

source and destination) dominate the total traffic volume [14]. This greatly *decreases* the entropy of all the address and port distributions.

Analyzing the flow records, we suspect that these arise due to botnet activity: compromised university hosts being used to attack external targets. In events $F - H$ and $L - N$, we observed a large volume of UDP traffic destined to a single external host on popular application ports (80,53). These flows are suspicious for three reasons. First, the direction and nature of the traffic is suspicious (normally in these applications we expect more traffic to client hosts, not from client hosts). Second, there is no response traffic from the destination host (it is either overwhelmed or it is dropping these packets). Third, we also found a small amount of TCP traffic on port 6667 (IRC) to and from the sources of the alpha flows just prior to the onset of the alpha flow. The IRC traffic suggests the activity is possibly botnet related (botnets commonly use IRC for command and control channels).

Additionally, eight of these alpha flow incidents share a source IP with another anomaly suggesting that the same compromised host was being used in multiple attacks. This has implications on detection as using entropy catches the hosts engaging in malicious behavior which removing them based on the alarms would prevent the future events seen sharing sharing infected hosts observed in the trace.

Event H is particularly interesting in terms of our heuristic for discovering anomalous flows, as it consisted of two independent alpha flow events. Our initial analysis revealed one alpha flow. However, after observing that the anomaly persisted after temporarily removing the initial alpha flow event, we discovered the second alpha flow as well.

In our dataset, the alpha flows are the *only* anomalies that any of the port and address distributions detect. Further, these anomalies are detected by *all* the port and address distributions. This implies, at least in our dataset, that using all the port and address distributions in conjunction provides no marginal benefits over using just one of these distributions. Not only do addresses and ports detect the events, but simple volume based detection could have been effective in detecting the alpha flows observed in the trace as each produces between a 25% and 200% increase in traffic. This implies reduced utility in monitoring ports and addresses, which will also seen in the synthetic anomalies.

Peer to Peer Supernode Activity: Removing the alpha flows for event N from the traffic does not eliminate a series of concurrent FSD anomalies (collectively labeled O). Further analysis of the FSD anomaly reveals that it was caused by an internal host being recruited as a "supernode" in the Kazaa network [10]. During the event, many hosts connect to this supernode creating a significant number of small flows, which sharply decreases the entropy of the FSD.

By monitoring just the port and address distributions it would not have been possible to determine that there are two separate anomalous events. This suggests the need to consider distributions that capture different structural properties of the underlying traffic.

Scan Activity: During event A, a single internal host scanned more than 350,000 unique external hosts. The scanner used a fixed source port of 666. The destinations ports of the scan are between 0–1024, and each of the 350,000 hosts is scanned exactly once. As there are a large number of small flows, FSD detects this scanning. The source address and port distributions do not detect the behavior, as the source does not generate enough packets to bias these distributions significantly.

Since only one internal host is involved, the degree distributions are unaffected.

Event D is a more traditional (outbound) port scan, with a single internal host scanning numerous external hosts on multiple ports. FSD alone detects the scan while the other traffic features remain unaffected.

Contrary to conventional wisdom, port and address distributions do not show significant deviations for the scanning anomalies in our data. FSD detects such abnormal scanning activity which we revisit using synthetic anomalies.

Possible DoS using spoofed addresses: In anomalies C, E, and I, a very large number of "hosts" (we speculate that these are spoofed) have out-degree 1. This decreases the entropy of the out-degree distribution (demonstrating its uniqueness and little overlap). The majority of the "hosts" with out-degree 1 connect to the same external destination on port 6667. The set of source addresses in these flows spans the entire /16 of the university address space. Oddly, the source ports of the flows fall within a small range of port numbers. Also, the destination host sends a single packet response, but the flows seem to terminate abruptly. This leads us to believe that an internal host may be sending attack traffic with spoofed source addresses (within the same subnet) to avoid egress filtering[4].

The presence of these anomalies unique to the out-degree reiterate the need to choose distributions that provide a different view in to the structure of flow level traffic.

3.5 Summary of Measurement Results

- The entropies of the port and address distributions (both source and destination) are strongly correlated.
- The behavioral distributions (in- and out-degree) and the FSD exhibit low correlations with each other and with the port and address distributions.
- The correlations in the deviation scores mirror the correlations in the entropy values using both anomaly detection methods.
- The anomalies detected by port and address distributions overlap significantly. Further analysis reveals that most of these are alpha flows.
- In contrast, the degree distributions and FSD detect unique anomalies that are not captured by other distributions. For example, FSD detects interesting scan and P2P behaviors, even when the anomalies are embedded in larger alpha flow anomalies.

3.6 Using Synthetic Anomalies

In this section, we use synthetically generated anomaly events to complement our measurement results. Table 3.7 presents a taxonomy of the different types of synthetic anomalies we evaluate.

[4]Since we only have anonymized flow level traces, we could not further validate this hypothesis.

Anomaly Type	SrcAddr	DstAddr	SrcPort	DstPort	FlowSize
Inbound DDoS Flood	Random	Fixed	Random	Fixed	Fixed (10 Kbps), 1 flow per packet
B/W Flood	Random	Fixed	Random	Fixed	Random (300-400 Kbps), 1 flow per host
Single Scanner	Fixed	Random	Random	Fixed	1-3 packets (10% response rate)
Multiple Scanners	Random	Random	Random	Fixed	1-3 packets (10% response rate)
Port Scan	Fixed/Random	Fixed	Random	Fixed	1-3 packets (10% response rate)

Table 3.7: Taxonomy of synthetic anomalies used in our evaluation

For each type of anomaly, we are interested in three questions:

1. Can the anomaly be detected using any of the feature distributions?

2. Which feature distribution is most effective for detection?

3. How sensitive is the detection to the magnitude of the anomaly?

To understand the sensitivity of the detection metrics we vary the scale of the anomaly using a specific control parameter (e.g., number of source hosts involved in a DDoS or scan attack). In the case of the DDoS and bandwidth floods, we are also interested in comparing the merits of entropy-based detection to simply volume based detection. Specifically, we want to identify the smallest anomaly for which entropy-based detection becomes feasible, and reason whether at this magnitude volume-based analysis would have sufficed.

We set the duration of the anomaly to be a single five-minute timeslot. For each experiment (i.e., given an anomaly type and the anomaly magnitude), we introduce the anomaly at 50 semi-randomly selected locations in the month-long trace (semi-randomly because we ensure that these locations do not overlap with previously discovered traffic anomalies in the dataset). Each data-point in the following results represents the mean anomaly deviation scores (using wavelet-based detection) over the 50 experiments; the standard deviations were small and not shown.

For brevity we only consider *inbound* anomalies: the source addresses involved in the incidents are outside the university and the destination addresses are inside the university. The results for outbound anomalies are qualitatively similar (except that the roles of in and out-degree get reversed).

3.6.1 Inbound DDoS Flood

Finding: *FSD detects DDoS floods with a 6× lower magnitude than port and address distributions.*

A DDoS event is characterized by a single destination address receiving a large volume of single-packet flows (to overwhelm both the bandwidth and processing capacity of the server and routers). Figure 3.5 shows the anomaly scores as a function of the number of hosts participating in the DDoS attack. Each attack source generates 10 kilobits per second of attack traffic, using a fixed packet size of 57 bytes and a single flow per packet. The attack flows are destined to port 80 on a randomly chosen host inside the university. We have repeated the experiments varying the destination port, and the choice of destination address (picking a high-volume, random, and low-volume host) and found similar results.

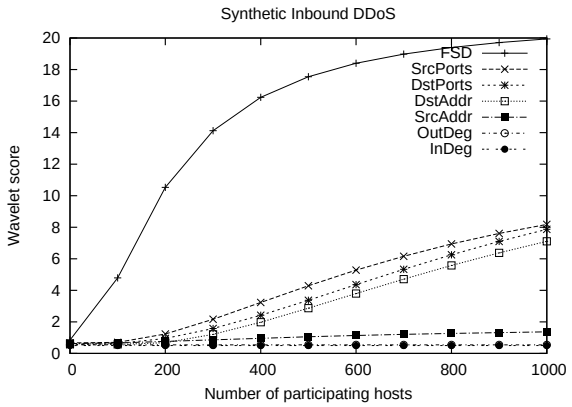

Figure 3.5: Anomaly scores for inbound DDoS flood

With just 100 hosts generating 10 kilobits per second of attack traffic each within a five minute interval, there would be $100hosts * 300sec * 10Kbps/57bytes \approx 650,000$ flows of size 1. Thus, the change in the FSD can easily detect the anomaly even with a small number of participating hosts. The destination port and destination address distributions detect the anomaly (with the score exceeding the threshold $\alpha = 3$) only when more than 600 attack sources are involved. The anomaly is also detected by source ports since the distribution of traffic across source ports appears random (with 600 hosts participating the normalized source port entropy exceeds 0.97). The degree distributions are unaffected by this anomaly.

3.6.2 Bandwidth Flood

Finding: *Destination port and address can detect the anomaly. But, the magnitudes of the anomalies that can be detected are large enough that simple volume-based analysis would suffice.*

In a bandwidth flood, a small number of high-bandwidth hosts send traffic to a single destination. The key differences with respect to the DDoS attack are that the number of hosts involved is an order of magnitude smaller than the DDoS, and that each attack flow is a single high-volume flow. (All the attack packets from a single source have the same source port and thus get aggregated into a single flow, as opposed to a DDoS attack where each packet has a random source port.)

We vary the number of hosts involved in flooding a single target IP address. The rate of traffic from each host varies uniformly in the range of 300 to 400 Kilobits per second with a fixed packet size of 57 bytes. The flood is targeted against a single destination host (chosen at random within the university) on a specific destination port (e.g., port 80 on a webserver). Again, the results were independent of the choice of port and destination host.

Figure 3.6 shows that the behavioral features, FSD, source port, and source address are unaffected. Since each source generates a single flow (and the size of each flow is random), FSD is not effective at detecting this anomaly. As expected, destination ports and addresses exhibit the greatest deviation. However, a simple back-of-envelope calculation suggests that traffic volume can detect this anomaly as well. With 40 hosts (the smallest magnitude that exceeds a detection threshold $\alpha = 3$ using destination ports) sending 300 Kilobits per second each, an additional 8 mil-

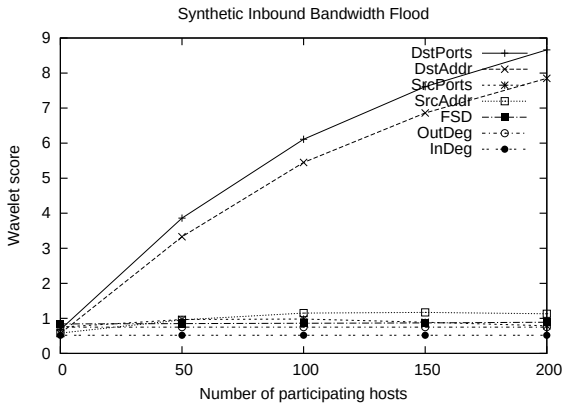

Figure 3.6: Anomaly scores for inbound bandwidth flood

lion packets are introduced. This is roughly 25% of the peak aggregate traffic volume we observe per-day. This suggests that this flood will be detected by simple volume-based anomaly detection as well.

3.6.3 Network Scans

Scan traffic is usually made up of a large number of flows to a specific destination port that is being targeted for exploits. We consider two types of scanning activity: a single scanning host trying to scan the entire university address space, and distributed scanning activity arising from a set of random source addresses. One of the reasons for selecting addresses and ports as candidate traffic features is that such scan traffic typically affects these distributions. However, we find that these metrics are ineffective for detecting scanning activity. In contrast, FSD and in-degree detect even low magnitude scanning activity.

We set the destination port for the scans to be 445 (associated with many known vulnerabilities [20]). The results are independent of the choice of destination port. To closely model the observed properties of real scanning activity (e.g., what fraction of hosts respond, what are the flow sizes of the probes and responses), we sample 10,000 inbound scan flows to port 445 from the traffic trace. We observe that scans receive responses to probes approximately 10% of the time, and that in these cases the probe flows have a flow size of 3 packets, else they are single-packet flows (just a SYN packet).

Single Scanner

Finding: *Entropy-based anomaly detection using traffic feature distributions cannot detect single scanners with scan rates less than 200 scans per second.*

Figure 3.7 shows that a host scanning at 200 hosts per second goes undetected with the entropy metrics. FSD shows the largest deviation, but even it does not generate an anomaly score above 1. Among the other distributions, we observe slight changes in the in-degree (the entropy decreases because there is a sudden increase in number of hosts with in-degree 1) and destination addresses

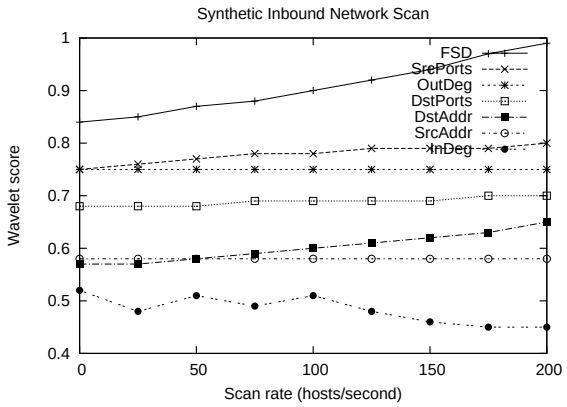

Figure 3.7: Anomaly scores for a network scan with a single scanning host

(the entropy increases since many hosts receive a small volume of traffic and the distribution becomes closer to being uniform). However, the deviations are not large enough for detection.

For detecting such isolated scan activity, more fine-grained per-host analysis (e.g., flag any host contacting more than X unique destinations in Y seconds [1]) and incorporating other aspects of scanning behavior (e.g., failed connections [8]) are necessary.

Multiple Scanners

Finding: *FSD and in-degree detect scan/worm activity with aggregate rates an order of magnitude lower than port and address distributions.*

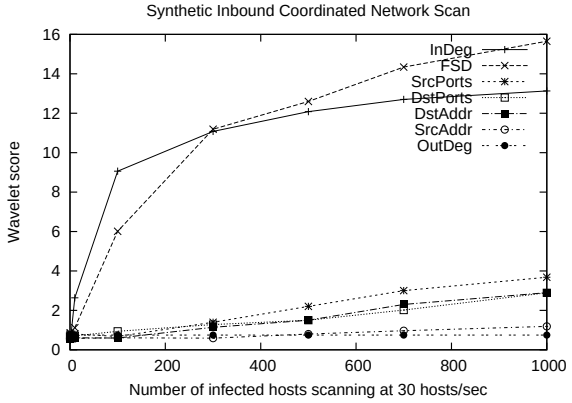

Figure 3.8: Anomaly scores with the network scan having multiple scanners

Such activity is representative of coordinated network scans and worms. In a coordinated scan, multiple hosts (e.g., part of a botnet) scan a particular network. Each participating host generates scan at a low rate – this can cover the entire address space quickly without incurring blacklisting (e.g., hosts generating more than a threshold X scans per second might be blacklisted by IDSes [1]). (The outbreak of worm activity produces identical behavior. With a random scanning

worm, the probability of an incoming scan is $InfectedHosts * ScanRate * \frac{InternalAddressSpace}{TotalIPAddressSpace}$.) We fix the scan rate to 30 hosts per second, and vary the number of hosts generating scanning activity. As in the single scanner case, we assume a 10% response rate.

Figure 3.8 shows that in-degree and FSD exhibit the strongest deviations. Even with a small number (less than 100) scanners, we observe anomaly scores greater than 9 and 6 using in-degree and FSD respectively. Intuitively, we expect the destination port distribution to be effective at picking such scanning activity since port is invariant across the scan traffic. Surprisingly, the destination port distribution does not detect the scanning activity even when there are 1000 scanners.

FSD and in-degree have another favorable property: they detect scans independent of the ports used by the scan traffic. This implies that even when the vulnerability exists on an application associated with a lot of normal traffic, FSD and in-degree still detect the abnormal scanning activity. However, the destination port distribution misses this scanning activity since the change in the traffic on the popular application port relative to the normal traffic will be negligible.

3.6.4 Port Scan

Finding: *FSD alone is effective for port scan detection.*

We also explore several synthetic port scans. The results are very similar to the case of network scans, with the only exception being that the degree-based metrics are ineffective. A single scanner with a moderate scan rate (30 scans per second) cannot be detected by any of the entropy metrics (as in Figure 3.7). With an increased scan rate (> 1000 scans per second) or with multiple scanners, FSD is the only metric that detects the port scan. The port, address, and degree distributions remain unaffected by the port scanning activity even with a large number of scanners or a high scan rate.

3.6.5 Summary of synthetic anomaly study

- FSD provides the best detection for DDoS attacks with many small flows.
- While destination port and destination address can detect bandwidth and DDoS floods, the attack magnitudes they detect are large enough to be detected with simple volume based detection.
- Single scanners are very hard to detect using entropy based metrics. Even with a scan rate more than 30 hosts per second (scanning a class B subnet in a half hour) the scan cannot be detected by entropy based anomaly detection.
- Port and address distributions are ineffective for detecting scanning anomalies. FSD and in-degree are the most promising metrics for scan/worm detection.

Chapter 4

Datapository Anomaly Detection Testbed

In this section, we present the Datapository Anomaly Detection Testbed (DP-ADT), a framework and storage facility for analyzing and developing detection methods, generating and labeling anomalies, and analyzing traffic features with user provided traffic sets or publicly available traffic sets in the Datapository database. The framework includes the detection methods, synthetic anomalies, and traffic analysis tools used to complete this work. Making the framework publicly available not only allows others to perform a similar study, it also allows users to expand the framework by adding newly proposed detection methods, traffic features, or synthetic anomalies.

4.1 Architecture

DP-ADT is constructed using a two-level architecture: a PostgreSQL [24] backend and Ruby [27] frontend. The PostgreSQL backend provides structured storage for the traffic and resulting meta data (e.g., entropy values, labels, deviation scores), as well as simplifies the computation of traffic features and generation of synthetic anomalies. The Ruby frontend provides the user a powerful and modular environment to develop and evaluate detection methods, label anomalies, generate synthetic anomalies, and create new functionality using provided core methods.

4.1.1 PostgreSQL Backend

The database structure used for DP-ADT is shown in Figure 4.1. The basic flow information is stored in two tables: *intervals* and *flows*. As all major auditing tools split traffic into intervals, the top most *intervals* table keeps track of the details of the intervals. The *intervals* table is also flexible to caching information in the future, such as the number of inbound or outbound flows. This table is referenced by several other tables as entropy values, deviation scores, and alarms belong to intervals and not flows. The *flows* table stores the raw flow information, the interval the flow belongs to, and a unique (per interval) flow identifier for labeling flows. IP addresses are stored in integer format as PostgreSQL has no unsigned type.

The *flows* table is partitioned by interval with consistency checks to ensure all flows belong within the interval according to timestamp. Partitioning the flows table is essential to the perfor-

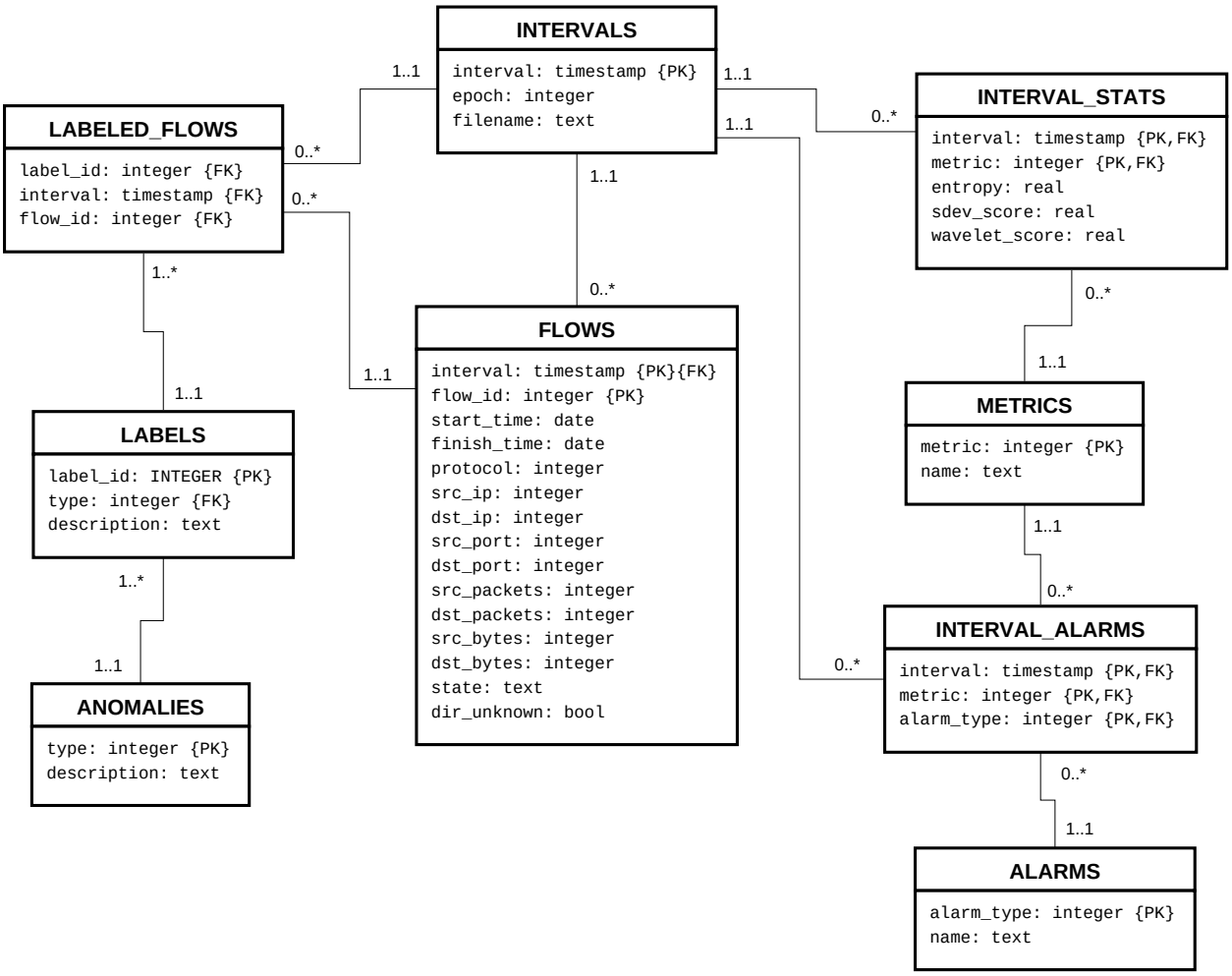

Figure 4.1: Database structure for DP-ADT.

mance of the framework. Given that almost all queries are run on single intervals, partitioning the flows by interval provides the greatest performance.

The database also supports caching of information through the tables *interval_stats* and *interval_alarms* which is crucial to the performance of the testbed. The tradeoffs of caching this information is discussed in Section 4.1.3. However, the *interval_stats* table stores cached statistics that are representative of an interval, such as entropy and deviation scores. As each traffic feature will generate different entropy values and deviation scores per interval, a metric value is stored with each set of statistics. The metrics are stored in the *metrics* table, which is also flexible to new traffic features.

Alarm caching is achieved in the database using the tables *alarms* and *interval_alarms*. The *alarms* table stores the different alarm types such as *wavelet-3*, which would be alarms generated using wavelet detection and a threshold $\alpha = 3$. The *interval_alarms* table stores the cached alarms for intervals based on the metric and alarm type.

Storing labels is done through the *anomalies*, *labels*, and *labeled_flows* tables. The *anomalies* table stores anomaly types, such as *inbound worm activity*. As anomaly events typically consist of multiple flows, the *labeled_flows* table references the flows which belong to an anomaly. Each labeled anomaly is stored in the *labels* table and has an associated anomaly type and an optional user description such as: *a local intranet host flooding an external host*. The details of labeling anomalies is further discussed in Section 4.2.3.

4.1.2 Ruby Frontend

The Ruby frontend provides an environment with core methods to interface to the PostgreSQL backend and also provides the flexibility of user defined methods to expand the functionality of the testbed. Ruby is a powerful object-oriented programming language which the core functionality is written in, independent of the underlying database. This allows the framework of the testbed to be reused on other databases and keeps compatibility in the event of a database change.

Interfacing with the backend is done using the *ruby-postgres* library [28] which provides methods for connecting to the backend and executing queries. Queries are written in SQL and encapsulated in a Ruby string, which is sent to the backend for execution. Generating the queries using Ruby strings also makes dynamic queries extremely simple, which when done in PostgreSQL would require an *EXECUTE* statement with complex and rather unreadable plpgsql code.

The results of the queries are returned to the Ruby environment as a two dimensional array, where the first dimension represents a row in the results and the second dimension represents the column. Errors are also reported by the library. After the results are returned, they can be easily manipulated and used to generate additional queries.

4.1.3 Caching

Caching is seen in the traffic analysis and anomaly detection components of DP-ADT to increase the performance and usability of the testbed. A value is cached if it is unchanging, used repeatedly, requires significant computation, and caching it would not incur large storage overhead. As an example, the entropy of an interval and metric does not change over time, is used several times in

anomaly detection, takes on the order of hours to compute for a month long trace, and caching the values for a month long trace requires under 1 megabyte of storage. Therefore, entropy values are cached using the *interval_stats* table.

Although deviation scores and alarms typically only take on the order of seconds to generate after computing entropy, they are used repeatedly and considered historic with the data. These values are also often studied with labels which are stored by DP-ADT and can be used in future studies of the traffic trace without the need for users to recompute the values.

Values which are not cached are the traffic feature statistics (e.g., port X sourced Y packets) due to the amount of storage required to cache them. Although the traffic feature statistics are arguably the greatest computational bottleneck in the testbed, they are rarely used more than once. The traffic feature statistics are used once for entropy computation and only used again if manual labeling on an interval is done. Considering manual labeling typically occurs on less than 5% of data points, will not occur in all data sets (those already labeled), and requires gigabytes of storage overhead, these values are not cached by DP-ADT. Furthermore, during labeling these results can be temporarily cached through variables in the Ruby environment.

4.1.4 Modularity

A goal in the design of DP-ADT is modularity in the Ruby frontend. Individual methods provide minimal and simple functionality that when combined perform more complex tasks. This increases the re-usability of many of the methods for different tasks. As an example, the raw traffic statistic methods discussed in Section 4.2.1 are combined with the entropy methods to compute normalized entropy. However, these raw traffic statistics are also used in manual labeling and traffic analysis.

DP-ADT provides a set of *core methods* which provide simple functionality and are used in nearly all components. The goal of these methods is to provide basic building blocks for more complex functionality. Examples of these methods are *get_metrics()* and *get_all_intervals()*, which retrieve all available metrics and the timestamps of all intervals respectively. When these methods are combined, the entropy of all metrics and intervals can be computed and cached:

```
get_metrics().each do |id,metric|
  get_all_intervals().each do |i,e|
    entropy = compute_entropy(i, metric)
    cache_entropy(i, metric, entropy)
  end
end
```

This goal should be kept in mind when expanding the functionality of DP-ADT. A subset of functionality should be removed from a method and placed in to a separate method if it can be used in other aspects of the framework.

4.2 Component Design

In the following section, we discuss the design of the core components of DP-ADT: traffic analysis, anomaly detection, labeling, and synthetic anomalies. The design of each component addresses performance, storage, and flexibility.

4.2.1 Traffic Analysis

The design of the traffic analysis framework in DP-ADT follows the architecture: perform the majority of computation on the backend where database queries simplify the process, and return the results to the Ruby frontend where the results can be used for further in depth study. The traffic analysis framework computes the raw traffic statistics (e.g., in-degree of host X, packets sourced from port Y) on the backend whose results are returned by Ruby methods generating the queries, and the rest of the functionality is written in Ruby.

Traffic Statistics: Computing the raw traffic statistics is done on the backend for simplicity and performance. All traffic statistics can be generated using a single SQL query, rather than dozens of lines of Ruby which takes over twice as long to run. This is a major benefit of keeping the raw traffic flows in a database. As discussed in Section 4.1.3, these values are not cached. Although the statistics are computed on the backend, Ruby methods are provided as a wrapper which generate the queries and return the results. The interface design to these methods is to accept a mandatory interval parameter to compute the aggregate statistics and an optional table name parameter which is essential to the synthetic anomaly design, further discussed in Section 4.2.4. The default table is the *flows* table.

Returning the results to the Ruby frontend allows the user to easily examine and study the results in a more powerful environment where they can be temporarily cached. These methods provide the functionality of the *top-k* labeling approach discussed in Section 3.4, which labels most anomalies within minutes. To illustrate the simplicity of generating and analyzing the raw traffic statistics with the use of the Ruby framework, we provide the following examples:

- Display the top 3 hosts in terms of in-degree:

```
irb> stats_addr_degree_in(``2005-02-01 00:00:00'').first(3).each {|h,d| puts ``#{h} #{d}''}
191102 5393
16085133 3855
180751 3309
```

- Display the out-degree of host 191102:

```
irb> stats_addr_degree_out(``2005-02-01 00:00:00'').assoc(191102).last
=> 62
```

- The number of hosts with an in-degree of 3:

```
irb> stats_degree_out(``2005-02-01 00:00:00'').assoc(3).last
=> 989
```

- The number of packets sourced from port 80:

```
irb> stats_ports_src(``2005-02-01 00:00:00'').assoc(80).last
=> 1769099
```

Computing and Caching Entropy: A goal of the framework design is to be modular, such that simple methods can be reused and combined to perform more complex tasks on the frontend. This is seen in computing and caching entropy values in DP-ADT. The methods used to generate raw traffic statistics on the backend are reused to compute the normalized entropy on the frontend in the following manner: *entropy_degree_in(stats_degree_in("2005-02-01 00:00:00"))*. The returned value is a float representation of the normalized entropy for the specified interval.

A second interface, which acts as a wrapper to computing entropy is provided, is simpler to use and more readable: *compute_entropy(interval,metric)*. This ensures that as new metrics are added to the framework the basic interface does not change and code remains compatible (e.g., the example given in Section 4.1.4). The method branches on the metric, computes the raw traffic statistics on the backend, and then uses the returned values from the backend to compute the entropy. Therefore, when a new metric is added a new branch must be added to this method to call the correct raw statistic and entropy method.

DP-ADT is designed such that caching values is done just as easily as computing them. As goals of the testbed are performance and re-usability, it is recommended that the entropy be cached. Caching takes place in the *interval_stats* table with *interval* and *metric* associations, as each interval and metric pair has a unique entropy value. Using these two fields as the primary key ensures flexibility as new metrics are introduced in to the framework, changes will not need to be made to the database architecture. Caching entropy is done using *cache_entropy(interval,metric,value)*, as seen in the example given in Section 4.1.4. Likewise, reading the cache is done using *read_entropy(interval,metric)*.

Correlating Data: Correlating data was used throughout our trace analysis to understand the structure of traffic features and the feature overlap in the anomaly space. DP-ADT provides the *correlate()* method and the Ruby script *gen_correlations.rb* to aid in data correlation. The *correlate(a1,a2)* method is used in conjunction with the core methods, which takes two arrays of arbitrary data as parameters and returns the correlation coefficient as a float. We provide the example of correlating the entropy between source and destination addresses below to further illustrate the simplicity of the frontend. The overlap of alarms between the metrics with a wavelet threshold $\alpha = 3$ could also be computed by replacing *read_entropy(i,"addr_src")* with *read_alarm(i,"addr_src","wavelet3")* in the example.

```
a1=Array.new
a2=Array.new

get_all_intervals().each do |i,e|
  a1.push(read_entropy(i,``addr_src'')
  a1.push(read_entropy(i,``addr_dst'')
end

correlate(a1,a2)
```

To simplify pairwise correlation of metrics, DP-ADT provides the *gen_correlations.rb* script which will run multiple correlations and output in a table format. The details of the script and its usage are provided in the DP-ADT wiki.

4.2.2 Anomaly Detection

A core component of DP-ADT is the anomaly detection capability. While DP-ADT currently provides wavelet detection and heuristic-threshold detection, both of which are described in Section 2.2, the framework is designed to support development and evaluation of new methods. By making each detection method simple to run, and through the labeling of anomalies described in the next section, DP-ADT creates an environment where detection methods can be compared and evaluated.

Detection Method Design: The design of the detection method framework is slightly different than that of the general architecture. Detection methods need not be written in Ruby, they are only required to have a standard Ruby interface. While Ruby provides a powerful frontend to the testbed, it is not sufficient for many anomaly detection tasks. For example, while wavelet detection only requires 25 lines of MATLAB [19], it would take a whole signal processing package to be written in Ruby. It is suggested that R (GNU S) [25] and Octave [22] be used for development of detection methods that require a heavy statistical components.

Regardless of the language, a Ruby method must exist for the detection method which acts as a wrapper to the detection method. For example, the wavelet and threshold-based detection methods take a hash with sortable keys representing the timestamps of the data, and returns a hash also indexed by timestamp with deviation scores as values. This ensures that the detection methods are easily accessible and compatible with the Ruby frontend. As an example, the current wavelet detection method is written in MATLAB, which is invoked by a Ruby method and returns the deviation scores to the frontend which can be cached:

```
entropy = Hash.new

get_all_intervals().each do |i,e|
  entropy[i] = read_entropy(i,``addr_dst'')
end

dev_scores = detection_wavelet(entropy)
cache_deviations(``wavelet_score'', ``addr_dst'', dev_scores)
```

Details of all detection methods and the caching of their scores can be found in the DP-ADT wiki. As seen in other examples, the *get_metrics()* core method can be used to generate and cache to deviation scores of all metrics.

Generating Alarms: Alarm generation is currently designed to be threshold based. Generating alarms is done through the *threshold_alarms(dev_scores, threshold)* method which returns a hash indexed by timestamp with binary values indicating alarm. The *dev_scores* parameter is a hash of deviation scores returned by a detection method and *threshold* is a float representation of the detection threshold α. This does not prevent detection methods from generating their own series of alarms based on parameters, it is only used as a post-processing method for threshold based detection.

Caching alarms is done using *cache_alarms(alarm_name, metric, alarms* where *alarm_name* is the name for the alarm type in the *alarms* table and *alarms* is an alarm hash return by *gen_alarms()*. We discuss the reason for caching alarms in detail within Section 4.1.3. If the alarm name and values already exist, the alarm values will be updated. The alarm name will be inserted in the to

alarms table if it does not exist. To read the alarms for all metrics or a specific metric during a specific interval, the *read_all_alarms(interval, alarm_name)* method can be used. The return type is a hash indexed by metric name with binary alarm values for the specified interval.

4.2.3 Labeling Anomalies

Labeled anomalies provide detailed insight on the detection capabilities of methods and metrics. DP-ADT is designed to support fully labeled data and the labeling of previously unlabeled data which provides insight in to false positive and negative rates of the detection methods. As more anomaly detection studies label previously unlabeled traffic sets, the labels will become accurate for determining false positive and negative rates on originally unlabeled data. The DP-ADT labeling framework is designed to support user descriptions, associations with flows, full label searching, and the extraction of labeled flows for in-depth study or synthetic anomaly generation.

Labeling and Typing: Anomalies are inserted in to the database referencing three types data: a description, flows, and an anomaly type. These three sets of data allow for detailed searching and studies of anomalies. The anomaly type groups all anomalies in to classes such as *inbound worm activity*, *outbound bandwidth flood*, or *inbound DDoS attack*. Requiring a class on anomalies allows users of DP-ADT to find specific anomaly types throughout all datasets. This is a key feature which allows users to perform an in-depth study on one specific anomaly type or extract specific anomalies and insert them as synthetic anomalies in other datasets for greater anomaly event coverage during a detection study. If an appropriate anomaly type does not exist during insertion, a new class can be created which ensures flexibility as new anomalies are discovered.

Anomaly classes are not meant to be specific, such as *slammer worm*. This would make typing anomalies difficult, as there are thousands of anomaly variations. When inserting an anomaly, the user is prompted for a description of the anomaly which provides an additional level of detail which can also be searched through. These descriptions should provide details of the *specific* event, such as the hosts involved, the type of worm, and why it is believed to belong to the specific anomaly class. Descriptions such as those provided in the detailed in-depth anomaly study in Section 3.4 are appropriate. All though extracting the associated flows with the anomaly can often provide this same information, it allows users to quickly find details of an anomaly.

Label Searching: Searching for anomalies can now take place in a two level hierarchy where users can search based on types and descriptions. Searching based on types is done using the unique type ID assigned and stored in the *anomalies* table. Description searching is done using keywords with the *search_labels_or()* and *search_labels_and()* methods, which search descriptions using a logical *or* or *and* on the set of keywords. An optional anomaly type can be provided as a second parameter to only keyword search within a specific type. The following are examples of searching labels:

```
irb> print_labels(search_labels_type(9))

Label: 1
Type: outbound alpha flow
Interval Span: 2
```

```
Attack flows: 4
Description:
- increased traffic on port 80 (magnitude larger than normal)
- destined to the single host: 402737628 (internet)
- 4 flow sizes > 755191 packets
- A single Intranet hosts generate all of this additional traffic: 179858
- All traffic is UDP

irb> search_labels_or(``InTrAnet slaMMer'').each {|l| print_labels(get_label(l))}

Label: 1
Type: outbound alpha flow
Interval Span: 2
Attack flows: 4
Description:
- increased traffic on port 80 (magnitude larger than normal)
- destined to the single host: 402737628 (internet)
- 4 flow sizes > 755191 packets
- A single Intranet hosts generate all of this additional traffic: 179858
- All traffic is UDP

Label: 2
Attack: outbound worm activity
Interval Span: 1
Attack flows: 5
Description:
- slammer worm activity from multiple internal hosts
- external hosts seem to be chosen randomly
- scan rate seems to be lower than 10 hosts per second
- more than 40 local hosts participating
```

Extracting Labeled Flows: The ability to extract labeled flows allows for in-depth studies at the flow level of anomaly types and synthetic insertion in to other datasets for detection studies. It is often that anomaly events in historical traffic sets will not cover all anomaly types, preventing detection studies on specific anomaly types within that traffic set. By allowing for extracting of labeled flows for synthetic anomaly generation, we provide the ability to insert real anomalous flows in to other datasets for studies.

The design of labeled anomalies to reference *interval* and *flow_id* fields in the *flows* table allows for easy extraction of labeled flows on the backend. The flows are then returned to the Ruby frontend where they can be studied in-depth or inserted in to synthetic flow tables using synthetic anomaly methods, further described in the next section and the DP-ADT wiki.

4.2.4 Synthetic Anomalies

Synthetic anomalies can provide a deeper understanding in to the detection capabilities of methods and metrics through generation and control of the magnitude of the anomaly. The Ruby frontend and PostgreSQL backend create a powerful environment for creating and generating synthetic anomalies through SQL queries. We have found that synthetic anomalies are accurately modeled in 2-3 lines of Ruby and a single SQL query using *generate_series()* and *random()*. The framework simplifies modeling of anomalies, allows labeled flows to be used in synthetic anomaly generation, and provides an environment where multi-dimensional anomaly spaces are possible.

Synthetic Architecture: A key problem that needs to be addressed when generating synthetic anomalies is that they are typically studied with background traffic such as user provided datasets, however these datasets will not be writeable to most users to insert the anomalies. To address this problem we create the notion of an *anomaly table* and leverage views to union the *anomaly table* with the *flows* table. All subsequent queries, such as entropy computation, are performed on the view using the optional table specific parameter in all traffic statistic methods. Not only does this address the access control problem, but it also allows users to easily study the traffic with and without the anomaly, without having to delete the synthetic flows. For example, after the synthetic flows are inserted in an *anomaly* table and a view is created to union it with the *flows* table, the user can examine the top five source ports with and without the anomaly as follows:

```
stats_ports_src(``2005-02-01 00:00:00'').first(5)                  # without anomaly

stats_ports_src(``2005-02-01 00:00:00'', ``anomalies'').first(5)    # with anomaly
```

Another problem we address using the *anomaly table* is performance ensuring consistency. Without the *anomaly table*, the synthetic flows would need to be generated and inserted in to the *flows* table carrying a mark to ensure they can be accurately removed. This would incur an additional field in the *flows* table, increasing the storage overhead of the system. Given the smallest representation in PostgreSQL is a 1 byte boolean, the 2.5 billion flow traffic set used in the trace analysis would incur a 2GB overhead to support synthetic anomalies.

The *anomaly table* and view also simplifies averaging synthetic results. Without the *anomaly table*, the flows must be moved from partition to partition in *flows* table in between computing each result. This method increases the chances of inconsistency in the dataset, as one can easily lose track of where anomalous flows are in the system. To move flows to a new interval for averaging, their interval timestamps only need to be modified in the *anomaly table*, they are then interpreted as existing in another partition and interval.

Modeling Synthetic Anomalies: Modeling synthetic anomalies is made simple through Ruby and PostgreSQL. In the architecture, most models can exist solely in a single SQL query with a Ruby wrapper. In SQL, the *generate_series()* function allows multiple flows to be creating with a single query, and using the *random()* function gives each flow variability (e.g., source port). To demonstrate this, we present the most complex synthetic anomaly, multiple scanners:

```
def insert_ib_mnscan(anomaly_table, interval, scanners, subnet_start, subnet_end, scan_rate, scan_port)

  num_flows=(scan_rate.to_i*300).to_i   # convert scan rate in seconds to 5 min (interval size)

  # each scanner host will add num_flows flows, scanning random targets in the subnet
  (1..scanners).each do |scanners|

    # pick a random scanner from the whole 2^32 space
    scanner=(-2147483648+rand() * (2147483648 + 2147483648)).to_i

    conn.exec("
      INSERT INTO #{anomaly_table}
      SELECT
        '#{interval}',                                          # interval
        NULL, NULL,                                             # start/end time
        6,                                                      # protocol
```

```
        #{scanner}, between(#{subnet_start},#{subnet_end}),     # src/dst IP
        between(1024,65535), #{scan_port},                      # src/dst port
        1+resp, resp,                                           # src/dst packets
        57+(57*resp), 57*resp,                                  # src/dst bytes
        'CON', false                                            # connection info
      FROM
        generate_series(1,#{num_flows}),                        # generate num_flows
        FLOOR(.10+random()) AS resp                             # response packet?
    ;")
  end
end
```

In the most complex case, the synthetic anomaly is generated in 3 lines of Ruby, and a single SQL statement. The parameters allow the user to specify the interval to which anomaly will belong, the number of scanners, the start and end of the "victim" subnet, a scan rate (scans/sec), and a scan port.

As the scan rate is in scans per second and the interval size is 300 seconds (5 minutes), given that each scan is a single flow, we generate the number of flows based on the rate. Then, each scanner whose address is selected randomly in the whole Internet space, will insert *num_flows* to the given interval. The *generate_series()* function is used to produce *num_flows* in the database with a single query, and the random() function is used to give each flow variability. The *between()* function is DP-ADT SQL defined to produce a random number between the first and second parameters. Using this, we generate a random destination IP within the specified subnet. The source port is chosen randomly between 1024 and 65535, and the scan port is user specified and fixed. The model produces a 10% chance of response from the victim of the scan, which is generated in the last line of SQL as *resp*. Therefore, the source and destination packets will be 1 and 0, or 2 and 1, depending on the response decision for the given flow. With 57 byte packets, the proper byte counts are also generated.

The example illustrates the simplicity of modeling synthetic anomalies, even in the most complex case of the anomalies presented in Section 3.6.

4.3 Summary of DP-ADT

- The PostgreSQL backend simplifies raw traffic statistics, synthetic anomaly generation, and provides structured storage of flows and meta-data.
- The Ruby frontend provides a powerful and modular environment in which the majority of the framework is written in.
- Caching entropy, deviation scores, and alarms increases the performance and usability of the testbed with a very small amount of storage overhead.
- DP-ADT provides a framework to develop and test new detection methods, independent of the language the detection method is written in.
- The framework for labeling anomalies takes labeled data and new labels for previously unlabeled data. Anomaly types, user descriptions, type and description searching, and asso-

ciations with flows allows provides the user a powerful environment for in-depth anomaly studies and synthetic anomaly generation.

- Synthetic anomalies can be easily generated and safely maintained in a database environment. Most anomalies can be generated using single SQL statements and 2-3 lines of Ruby.

Chapter 5

Related Work

Network anomaly detection is a broad area of active research. Well known techniques for time-series analysis include the use of wavelets [3], time-series forecasting [26], and other signal processing techniques [32]. These techniques focus primarily on detecting deviations from an expected norm and are basic building blocks for several anomaly detection schemes.

The use of entropy and distributions of traffic features has recently received a lot of attention in the research community. Feinstein et al. [6] consider the use of entropy of the distribution of source addresses seen at a network ingress point for DDoS detection. Lakhina et al. [14] augment their PCA framework with entropy based metrics and show that this detects anomalies that cannot be identified using volume based analysis alone. These approaches show the promise of entropy-based anomaly detection. Our work seeks to better understand the selection of traffic feature distributions for entropy based anomaly detection. Since worm payloads contain common substrings that appear frequently, Karamcheti et al. [9] use distributional analysis over packet contents to detect malicious behaviors. We focus primarily on traffic features that can be inferred from flow level information.

In a much broader context, many traffic monitoring and intrusion detection applications use entropy as a measure of information. Lee and Xiang [16] propose information-theoretic measures for intrusion detection. Entropy has also been used in other contexts in network diagnosis in automatically clustering traffic patterns [36] and for analyzing the effectiveness of network monitoring solutions [17]. Wagner et al. [34] propose using the entropy for fast detection of worm attacks by evaluating the compressibility of flow data under worm attacks.

Many tools [33, 31] and generative models [11] attempt to synthetically replicate traffic structure in generating realistic traffic workloads for simulation and testing. The correlations we discover between the entropy values of the port and address distributions, both in the university dataset and in the Internet2 dataset, may have additional implications for such studies that aim to understand structural properties of IP traffic.

There has also been interest in specific algorithms for estimating distributional features at line-rates. These include the work on identifying the flow size distribution [13], and recent work on streaming algorithms for entropy computation [15]. In a related context, Brauckhoff et al. [4] evaluate how packet sampling affects the fidelity of entropy based anomaly detection, and show that sampling does not affect the accuracy of detecting the Blaster worm [20]. In a separate study, Mai et al. [18] suggest that packet sampling degrades performance significantly for anomaly detection.

These results are orthogonal to our measurement study as our focus is on better understanding the selection of traffic feature distributions for entropy based anomaly detection.

Other research efforts in the space of fine-grained anomaly detection have focused on measurements and analysis of backscatter and honeypot data [37], identifying heavy-hitters [5, 38], fast scanner detection [8] etc. These techniques are complementary to entropy-based analysis of flow level measurements.

Chapter 6

Conclusions

There is an evident need for fine-grained traffic metrics for anomaly detection and other traffic classification applications. Entropy-based metrics have been suggested as possible candidates to fulfill this need. The goal of this measurement study was to better understand the anomaly detection capabilities provided by different entropy based metrics.

We evaluated two classes of entropy based metrics: five based on flow header features and two based on properties of end-host behavior. We find that the port and address distributions are strongly correlated not only in their detection capabilities, but also in the raw values themselves. We also observe that the behavioral metrics (in- and out-degree) and the flow size distribution provide detection abilities that are distinct from other distributions. Using synthetically generated anomalies further confirms that the port and address distributions have limited utility in anomaly detection: they are ineffective for scanning anomalies, and the flood anomalies they detect are large enough to show as volume anomalies.

Our results have two significant implications in the context of entropy-based anomaly detection. First, we should look beyond simple port and address based traffic distributions for fine-grained anomaly detection. Second, we should consider sets of distributions that provide the greatest benefit benefit when used in conjunction with each other. One direction of future work in this context is to explore information-theoretic measures [12] to decide which sets of traffic features are likely to provide the best detection capabilities.

Bibliography

[1] Snort. http://www.snort.org.

[2] Argus. http://qosient.com/argus/.

[3] P. Barford, J. Kline, D. Plonka, and A. Ron. A signal analysis of network traffic anomalies. In *Proc. of IMW*, 2002.

[4] D. Brauckhoff, B. Tellenbach, A. Wagner, A. Lakhina, and M. May. Impact of traffic sampling on anomaly detection metrics. In *Proc. of ACM/USENIX IMC*, 2006.

[5] C. Estan, S. Savage, and G. Varghese.

[6] L. Feinstein, D. Schnackenberg, R. Balupari, and D. Kindred. Statistical Approaches to DDoS Attack Detection and Response. In *Proc. of DARPA Information Survivability Conference and Exposition*, 2003.

[7] Internet2. http://www.internet2.edu.

[8] J. Jung, V. Paxson, A. W. Berger, and H. Balakrishnan. Fast Portscan Detection Using Sequential Hypothesis Testing. In *Proc. of the IEEE Symposium on Security and Privacy*, 2004.

[9] V. Karamcheti, D. Geiger, Z. Kedem, and S. Muthukrishnan. Detecting malicious network traffic using inverse distributions of packet contents. In *Proc. of ACM SIGCOMM MineNet 2005*, 2005.

[10] Kazaa. www.kazaa.com.

[11] E. Kohler, J. Li, V. Paxson, and S. Shenker. Observed Structure of Addresses in IP Traffic. In *Proc. of IMW*, 2002.

[12] D. Koller and M. Sahami. Toward optimal feature selection. In *Proc. of ICML*, 1996.

[13] A. Kumar, M. Sung, J. Xu, and J. Wang. Data streaming algorithms for efficient and accurate estimation of flow distribution. In *Proc. of ACM SIGMETRICS*, 2004.

[14] A. Lakhina, M. Crovella, and C. Diot. Mining anomalies using traffic feature distributions. In *Proc. of ACM SIGCOMM*, 2005.

[15] A. Lall, V. Sekar, J. Xu, M. Ogihara, and H. Zhang. Data streaming algorithms for estimating entropy of network traffic. In *Proc. of ACM SIGMETRICS*, 2006.

[16] W. Lee and D. Xiang. Information-theoretic measures for anomaly detection. In *Proc. of IEEE Symposium on Security and Privacy*, 2001.

[17] Y. Liu, D. Towsley, T. Ye, and J. Bolot. An information-theoretic approach to network monitoring and measurement. In *Proc. of IMC*, 2005.

[18] J. Mai, C.-N. Chuah, A. Sridharan, T. Ye, and H. Zang. Is sampled data sufficient for anomaly detection. In *Proc. of ACM/USENIX IMC*, 2006.

[19] Matlab. `http://www.mathworks.com/`.

[20] J. Morrison. Blaster revisited. ACM Queue vol. 2 no. 4, June 2004.

[21] Cisco Netflow. `http://www.cisco.com/warp/public/732/Tech/nmp/netflow/index.shtml`.

[22] Octave. `http://www.gnu.org/software/octave/`.

[23] P. Phaal, S. Panchen, and N. Mckee. InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks .

[24] Postgresql. `http://www.postgresql.org/`.

[25] The r project for statistical computing. `http://www.r-project.org/`.

[26] M. Roughan, A. Greenberg, C. Kalmanek, M. Rumsewicz, J. Yates, and Y. Zhang. Experience in Measuring Internet Backbone Traffic Variability:Models, Metrics, Measurements and Meaning. In *Proceedings of International Teletraffic Congress (ITC)*, 2003.

[27] Ruby programming language. `http://www.ruby-lang.org/`.

[28] Ruby postgresql libary. `http://ruby.scripting.ca/postgres/`.

[29] V. Sekar, N. Duffield, K. van der Merwe, O. Spatscheck, and H. Zhang. LADS: Large-scale Automated DDoS Detection System. In *Proc. of USENIX ATC*, 2006.

[30] C. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, July 1948.

[31] J. Sommers and P. Barford. Self-configuring network traffic generation, 2004.

[32] M. Thottan and C. Ji. Anomaly Detection in IP Networks. *IEEE Trans. on Signal Processing*, 51(8):2191–2204, Aug. 2003.

[33] K. Vishwanath and A. Vahdat. Realistic and responsive network traffic generation. In *Proc. of ACM SIGCOMM*, 2006.

[34] A. Wagner and B. Plattner. Entropy Based Worm and Anomaly Detection in Fast IP Networks. In *14th IEEE International Workshops on Enabling Technologies, Infrastructures for Collaborative Enterprises (WET ICE 2005)*, 2005.

[35] J. Xu, J. Fan, M. H. Ammar, and S. B. Moon. Prefix-preserving IP Address Anonymization: Measurement-based Security Evaluation and New Cryptography-based Scheme. In *Proc. of IEEE ICNP*, 2002.

[36] K. Xu, Z. Zhang, and S. Bhattacharyya. Profiling internet backbone traffic: Behavior models and applications. In *Proc. of ACM SIGCOMM*, 2005.

[37] V. Yegneswaran, P. Barford, and J. Ullrich. Internet intrusions: Global characteristics and prevalence. In *Proc. of ACM SIGMETRICS*, 2003.

[38] Y. Zhang, S. Singh, S. Sen, N. Duffield, and C. Lund. Online detection of hierarchical heavy-hitters. In *Proc. of IMC*, 2004.