

# Detection and Identification of Network Anomalies Using Sketch Subspaces

Xin Li<sup>\*</sup> Fang Bian<sup>\*</sup> Mark Crovella<sup>†</sup> Christophe Diot<sup>‡</sup>  
Ramesh Govindan<sup>\*</sup> Gianluca Iannaccone<sup>§</sup> Anukool Lakhina<sup>†</sup>

## ABSTRACT

Network anomaly detection using dimensionality reduction techniques has received much recent attention in the literature. For example, previous work has aggregated netflow records into origin-destination (OD) flows, yielding a much smaller set of dimensions which can then be mined to uncover anomalies. However, this approach can only identify which OD flow is anomalous, not the particular IP flow(s) responsible for the anomaly. In this paper we show how one can use random aggregations of IP flows (i.e., sketches) to enable more precise identification of the underlying causes of anomalies. We show how to combine traffic sketches with a subspace method to (1) detect anomalies with high accuracy and (2) identify the IP flows(s) that are responsible for the anomaly. Our method has detection rates comparable to previous methods and detects many more anomalies than prior work, taking us a step closer towards a robust on-line system for anomaly detection and identification.

## Categories and Subject Descriptors

C.2.3 [Computer-Communication Networks]: Network Operations

## General Terms

Management, Security

<sup>\*</sup>Computer Science Department, University of Southern California, Los Angeles, CA 90089, USA. Email: {xinli, bian, ramesh}@usc.edu

<sup>†</sup>Computer Science Department, Boston University, Boston, MA 02215, USA. Email: {crovella, anukool}@cs.bu.edu

<sup>‡</sup>Thomson Paris Research Lab, 46 Quai A. Le Gallo, 92648 Boulogne Cedex, France. Email: christophe.diot@thomson.net

<sup>§</sup>Intel Research, 15 JJ Thomson Avenue, Cambridge CB3 0FD, UK. Email: gianluca.iannaccone@intel.com

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

IMC'06, October 25–27, 2006, Rio de Janeiro, Brazil.

Copyright 2006 ACM 1-59593-XXX-X/06/0010 ...\$5.00.

## Keywords

*Defeat*, Anomaly Detection, Sketches, Subspace Method

## 1. INTRODUCTION

Networks today increasingly see unusual traffic patterns. These traffic patterns arise from network abuse such as DDoS, port scans, and worms as well as from legitimate activity such as transient changes in customer demand, flash crowds, or occasional high-volume flows. These *traffic anomalies* are often difficult to detect at a single link and require scrutiny of the entire network [7].

Operators seeking to understand and manage their networks are increasingly looking at the details of network-wide traffic patterns using tools such as netflow. A central problem in this approach is dealing with the high dimensionality of traffic measurements. In particular, IP flows (flows indexed by 5-tuples) reside in a space of nearly  $2^{100}$  dimensions; characterizing “normal” traffic using this representation is intractable.

The recent research literature has proposed more tractable techniques for anomaly detection and classification [8, 7, 6, 12]. These proposals rely on a common approach to data analysis: they apply dimensionality reduction techniques such as sketches [12, 6] or principal components [7, 8] to the aggregate network traffic. Dimensionality reduction enables computationally efficient methods for identifying outliers (or anomalies) in the data set.

Most relevant to our work is that of [8], which addresses this problem by aggregating netflow into origin-destination (OD) flows, yielding a much smaller set of dimensions which can then be mined (using, for example, the subspace method) to uncover anomalies. However, this approach can only identify which OD flow is anomalous; the particular IP flow(s) responsible for the anomaly cannot be identified without manual examination.

In this paper, we show how one can use random aggregations of IP flows (*sketches*) to enable more precise identification of the underlying causes of anomalies. Our technique, called *Defeat*, is based on the novel insight that sketching the global traffic preserves the normal variation of flow traffic (the normal subspace), and most of the residual subspace. *Defeat* builds multiple sketches of feature entropies of the global traffic, then applies the subspace method of [8] to each of the sketches. Since each sketch randomly *shuffles* anomalies across different sketch entries, approximate agreement among sketches can be used to robustly detect anomalies. Accordingly, *Defeat* applies a voting procedure to the detection results from each of the sketches, which increases the

detection rate while reducing false alarms. Sketch entries are keyed using flow source or destination IP addresses, or combinations thereof; these keys are used to identify the IP-flows responsible for the anomaly.

We have evaluated *Defeat* on a week-long trace of flow records from the Géant and Abilene backbones, the same data sets analyzed in [8]. We find that *Defeat* is able to accurately pinpoint flows responsible for an anomaly. In addition, it has a detection rate comparable to [8] and catches many more anomalies than the method of [8]. We evaluate *Defeat*'s performance under different sketch sizes, numbers of sketches, and subspace sizes. This examination gives us some insight into why the method works well, by showing that random projections appear to preserve properties of traffic data that are important for the effectiveness of the subspace method.

*Defeat*'s use of multiple sketches takes us closer to an important goal: a robust on-line system for detection and identification of network anomalies. Computing the sketches, and applying the subspace method on them, are inherently parallelizable tasks and can be distributed across network elements, enabling robustness to failure and reducing computation and communication hotspots. Voting on the sketch results enables a higher detection rate and a lower false alarm rate. Finally, multiple sketches enable the automated identification of the IP-flows responsible for an anomaly. We intend to explore on-line anomaly detection and identification in future work.

## 2. DEFEAT

*Defeat*, like [8], is concerned with detection of network anomalies that are visible as unusual distributions of traffic features. A traffic feature is an entry in a packet header field; in this work we concentrate on source address (SIP), destination address (DIP), source port (SP), and destination port (DP). We use the entropy of the empirical distribution of each feature to detect unusual traffic patterns. We work with netflow traces, which are records of *IP flows*; for our purposes, each IP flow is a record consisting of  $\mathcal{T} = \langle F, P, T \rangle$  which denote the fact that a flow with 5-tuple  $F$  was present during the timebin  $T$ , and the flow consisted of  $P$  packets.

Like [8], we seek to *detect* time points at which unusual traffic feature distributions arise. However, *Defeat* attempts to go beyond previous work by also *identifying* which IP flow or flows are responsible for the anomaly.

To do so, we use sketches. There are many slightly different definitions of sketches, but the general notion is that of a random projection or aggregation of a high dimensional object into a smaller set of dimensions. The sense we mean here is random aggregation of IP flows.

Specifically, given a netflow record  $\langle F, P, T \rangle$ , a sketch of size  $s$  is constructed as follows. Select a hash function  $h(\cdot)$  which maps tuple values  $F$  to numbers in  $1, \dots, s$  (typical sketch sizes range from  $s = 32$  to  $s = 1024$ ). Then aggregate all IP flows which hash to the same value. The result is  $s$  'sketch flows' which can be used as input to anomaly detection algorithms.

The intuition behind random aggregation of IP flows is as follows. The mapping of IP flows to OD flows is just one particular mapping. If we instead randomly aggregate IP flows, there are two beneficial results: first, anomalies which are hidden under some aggregations will be exposed under other aggregations, leading to a higher detection rate.

Second, detections that only occur under a small subset of aggregations, but not under most aggregations, are likely to be false alarms and can be discarded. We demonstrate the effectiveness of these ideas later in the paper.

The second benefit of using sketches is that we can use multiple sketches in combination to *identify* anomalous IP flows. The general idea follows the multistage filters of [2], although with significant differences. Assume some collection of IP flows  $X$ . Construct  $m$  different sketches  $S_1(X), \dots, S_m(X)$  by using  $m$  different hash functions  $h_1(\cdot), \dots, h_m(\cdot)$ . Assume that we have detected an anomaly, and for each sketch we have determined the bin or bins which contain the offending IP flows. That is, in sketch  $S_1(X)$ , we know that bin  $f_1$  contains the offending IP flow, and so on. Then we can identify the IP flow with high probability by testing each tuple  $F$ , and observing whether  $h_i(F) = f_i$ , for all  $i = 1, \dots, m$ . Under suitable conditions on the hash functions  $h_1(\cdot), \dots, h_m(\cdot)$ , we can be confident that the IP flow(s) meeting this condition are likely to be responsible for the anomaly.

### 2.1 Defeat in Detail

Based on the general background presented in the last section, we now present the *Defeat* approach in detail.

Our setting is a network with  $N$  access points (ingress routers or PoPs)  $R_1, \dots, R_N$ , which we will generically call *routers*. Time is divided into discrete intervals or *time bins*. We use a time bin size of 5 minutes, which is consistent with prior work in the anomaly detection literature. In the rest of this section, we describe the algorithm as applied to a single time bin.

*Defeat* uses  $m$  hash functions  $h_1(\cdot), \dots, h_m(\cdot)$ . We use a 4-universal family of hash functions [10] to approximate a  $k$ -universal family of hash functions. For  $k$ -universal hash functions, the probability that two different keys both hash to the same value for each of  $k$  hash functions is exponentially small in  $k$ .

All sketches have the same size  $s$ , which is a tunable parameter; we explore the effects of varying  $s$  in Section 3.

In practice we do not hash the IP flow 5-tuple directly, but rather a function of it which we call the *hash key*. In our work, we use the first 21 bits of the source address concatenated to the first 21 bits of the destination address as the hash key. This tends to place IP flows that have 'nearby' sources and destinations into the same hash bucket. We note that other choices may be useful – for example, source and destination ports, or different address prefix lengths; we examine some of these choices in Section 3.

In the following we assume functions *SourceAddress*( $\cdot$ ), *DestAddress*( $\cdot$ ), *HashKey*( $\cdot$ ), etc., which extract the relevant parts of a IP flow's 5-tuple.

The *Defeat* algorithm proceeds in the following steps:

**1. Compute Local Sketches.** Each router  $R_i, i = 1, \dots, N$  collects the netflow records of the traffic entering the network at that point. Using these it forms  $m$  sketches for each of the four flow features (SIP, DIP, SP, and DP). That is, each node constructs  $4 \times m$  sketches.

As in [8], we use entropy to measure anomalous distributions. To enable this, router  $R_i$  maintains  $4 \times m$  histogram-sketches,  $m$  sketches for each of SIP, DIP, SP, and DP. Each histogram-sketch has  $s$  entries. For each  $j, j = 1, \dots, m$ , flow  $\mathcal{T} = \langle F, P, T \rangle$  is hashed to  $k = h_j(\text{HashKey}(F))$ , then a

record  $\langle \text{SourceAddress}(F), P \rangle$  is added to the SIP histogram-sketch  $j$ , entry  $k$ . The corresponding records are also added to the DIP, SP, and DP histogram-sketches. Thus for each router  $R_i$  there are histogram-sketches  $\mathcal{H}_{i,f,j}$ ,  $f \in \{\text{SIP, DIP, SP, DP}\}$ ,  $j = 1, \dots, m$ .

**2. Compute Global Sketches.** In the second step, the local histogram-sketches are summed to form global sketches. That is, for each  $f$  and  $j$ , we merge  $\mathcal{H}_{i,f,j}$ ,  $i = 1, \dots, N$  to form  $\mathcal{H}_{f,j}$ .

For each timebin  $T$ , the histogram contained in sketch  $\mathcal{H}_{f,j}$  can be treated as an empirical distribution. We then compute the entropy of this distribution. The result is a  $t \times s$  data matrix  $X_{f,j}$  which is suitable for input to the subspace method. There is one such matrix for each combination of feature and hash function.

**3. Detect Anomalies.** In the next step of the algorithm, the multiway subspace method [8] is applied to each  $X_{f,j}$ ,  $j = 1, \dots, m$ . For each hash function  $j$ , the method potentially registers a anomaly detection.

The subspace method works by separating the high dimensional space of traffic measurements (in our case,  $s$ -dimensional) into two subspaces which capture normal and anomalous variation, respectively. The key to the success of our method in this step is the empirical observation that random flow aggregation does not significantly disrupt the variation in traffic properties that falls in the normal subspace. We will discuss this observation in Section 3.

The outcome of this step is, conceptually, a bit vector  $b$  whose  $j$ -th bit is 1 if the subspace method detected an anomaly in the current time bin using hash function  $j$ , and 0 otherwise. As mentioned above, this bit vector  $b$  can be used to filter false alarms and increase confidence in true detections. To do so, we use voting approaches: we signal a detection if  $n$  out of  $m$  bits are set in  $b$ ,  $1 \leq n \leq m$ .

The choice of  $n$  is another tunable parameter. Choosing  $n$  equal to  $m$  is not necessarily the best choice. It can lead to a low false alarm rate, since if all  $m$  sketches agreed on the existence of an anomaly, it is highly likely that an anomaly actually exists. However, this strategy can result in missed alarms; since a sketch is a random projection of the global traffic, one or more sketches might fail to signal a detection, even when a true anomaly exists. We examine this trade-off in Section 3.

**4. Identify Anomalies.** Once an anomaly is detected via voting, the final step is to identify the IP flows(s) that contribute to the anomaly. The first task in doing this is to identify the sketch entries in each  $X_j$  that are anomalous; we use the greedy identification heuristic described in [8] to determine the anomalous sketch entries in each  $X_j$ . We then determine the set of keys that were mapped to the anomalous sketch entries. These keys are obtained from  $\mathcal{H}_{f,j}$ ,  $j = 1, \dots, m$ .

The intersection of the key sets over all hash functions  $j$ , which has raised the alarms, identifies the keys of the IP flows that caused the anomaly (with high likelihood). This step is the essence of *Defeat*'s identification procedure, and relies on the properties of  $k$ -universal hash functions, as described above. Finally, we output all IP flows whose 5-tuples match the chosen hash keys.

## 2.2 Related Work

Most of the early research in anomaly detection has focused on finding unusual changes in the overall traffic volume (i.e., byte and packet counts). Several schemes proposed in the literature are derived from classical time series forecasting and outlier analysis methods and applied to the detection of anomalies or faults in networks [3, 4, 5, 11]. More recently, these approaches have been extended to operate on flow measurements using wavelets [1] or Kalman filters [9]. A more recent thread of research has focused on methods that reduce the dimensionality of the original data set before applying classical change detection techniques [6, 12, 7].

An alternative method to dimensionality reduction has been discussed in [7]. Principal component analysis (PCA) is used in that work to separate the high-dimensional space occupied by a set of network traffic measurements into two disjoint subspaces corresponding to normal and anomalous network conditions. The main advantage of this approach is that it exploits correlations across links to detect network-wide anomalies. In [8], the authors extend this work by using PCA to detect anomalies in the sample entropy of the distribution of specific traffic features. *Defeat* is most closely related to [8], but differs in one extremely important respect: its ability to automatically drill-down to the IP-flows responsible for an anomaly. It also differs in several other respects. In *Defeat*, the dimensionality of the lower-dimensional object can be varied, and this can affect the detection rate, as we show below. Moreover, *Defeat* permits more flexible aggregation of flows than merely into O-D flows, and does not require additional information (routing tables and topology information) to compute these aggregates.

## 3. EVALUATION

In this section, we evaluate the performance of *Defeat*, and explore its parameter space. All our evaluations use the same data sets as in [8], the sampled netflow records from the Abilene backbone (December 15, 2003 to December, 21 2003), and from the Géant/Dante backbone (November 22, 2004 to November 28, 2004).

**Justification.** We begin by empirically demonstrating that random flow aggregation using sketches does not significantly disrupt the variation in traffic properties that falls into the normal subspace. Figure 1 plots the cumulative distribution of traffic variance as a function of the number of principal components (a scree plot [7]). We see that the scree plots obtained from random flow aggregation using three different hash functions is qualitatively similar to that obtained by aggregating the total traffic into O-D flows.

In addition, different random aggregates have qualitatively similar normal traffic characteristics. As Figure 2 shows, the first four eigenflows from two different sketches look similar; we have empirically verified this for all the hash functions used in this paper, but omit those graphs for brevity. This observation implies that the subspace method can be independently applied to the sketches; thus, this subspace computation in *Defeat* is inherently parallelizable, enabling a distributed implementation of *Defeat*.

**Detection Performance.** We now compare the performance of *Defeat* with that of [8]. For this, we choose the *Defeat* parameters shown in Table 1. We justify the choice

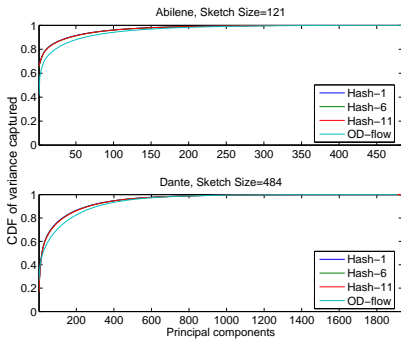


Figure 1: The scree plot of sketches and O-D flows. Sketch sizes are equal to the O-D flow numbers.

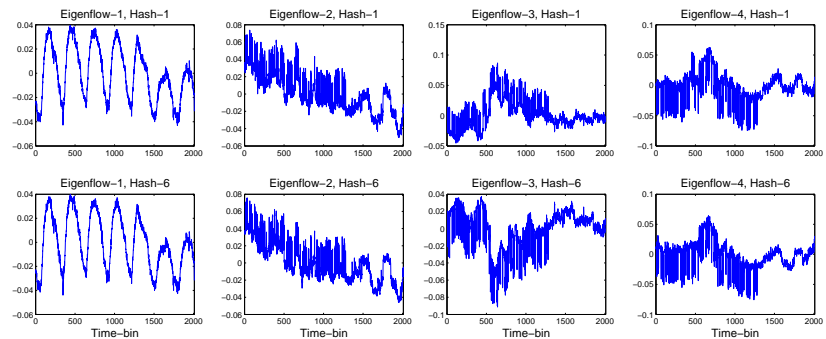


Figure 2: The first 4 eigenflows of the different sketches of Dante traffic. Sketch size is 484, equal to the O-D flow number.

| Parameters               | Values                     |
|--------------------------|----------------------------|
| Timebin size             | 5 min                      |
| Sketch size ( $s$ )      | 121 (Abilene), 484 (Dante) |
| Hash func number ( $m$ ) | 11                         |
| Normal space size (pdim) | 10                         |
| Key                      | source/21 + dest/21        |

Table 1: Parameters for evaluating detection performance.

|                              | Abilene | Dante |
|------------------------------|---------|-------|
| Lakhina <i>et al.</i> 's [8] | 90      | 161   |
| <i>Defeat</i> 's additional  | 14      | 192   |
| <i>Defeat</i> 's missed      | 32      | 25    |

Table 2: Detection performance compared with [8]'s result on the same data sets.

of most parameters later, but note that our choice of sketch size ( $s$ ) matches the number of O-D flows in Abilene and Dante.

We are interested in two measures of comparison. *Additional detections* are those reported by *all* of our hash functions, but not by [8]. *Missed detections* are those reported by [8], but *not* by *any* of our hash functions.

**Additional Detections.** Table 2 shows that *Defeat* detects nearly 200 more anomalies in the Dante data set than [8]. We manually examined *Defeat*'s additional detections for both data sets, and verified that *every one of them was an identifiable anomaly and not a false alarm*. Table 3 classifies these additional detections by anomaly type. We noticed that there were two kinds of additional detections: anomalies not picked out by [8] at all, and long-lasting anomalies that [8] picked out in some time bins and not in others. Why is *Defeat* able to detect many more anomalies? While

|             | Abilene | Dante |
|-------------|---------|-------|
| alpha flow  | 14      | 124   |
| DoS or DDoS | 0       | 32    |
| port scan   | 0       | 15    |

Table 3: *Defeat* additional detections.

|                            | Abilene | Dante |
|----------------------------|---------|-------|
| Not an anomaly             | 6       | 6     |
| Caught with different keys | 15      | 11    |
| Couldn't catch             | 11      | 8     |

Table 4: Defeat missed detections.

we don't have a rigorous explanation for this, we believe that since *Defeat* computes many different random aggregates of the traffic, it is able to catch anomalies that are hidden within the

**Additional Detections.** Table 2 shows that *Defeat* detects nearly 200 more anomalies in the Dante data set than [8]. We manually examined *Defeat*'s additional detections for both data sets, and verified that *every one of them was an identifiable anomaly and not a false alarm*. Table 3 classifies these additional detections by anomaly type. We noticed that there were two kinds of additional detections: anomalies not picked out by [8] at all, and long-lasting anomalies that [8] picked out in some time bins and not in others. Why is *Defeat* able to detect many more anomalies? While we don't have a rigorous explanation for this, we believe that since *Defeat* computes many different random aggregates of the traffic, it is able to catch anomalies that are hidden within the single OD flow aggregate computed by [8].

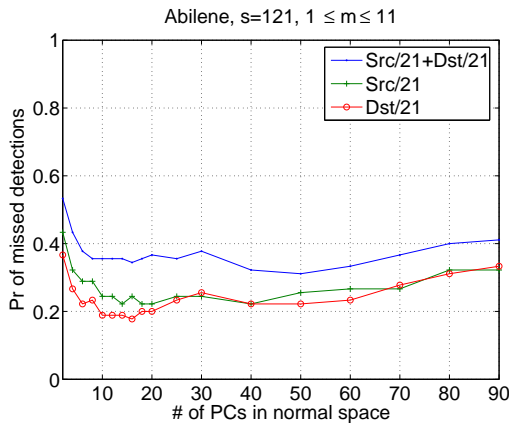
**Missed detections.** We also manually examined each of the missed detections, and classified them as shown in Table 4. First, six of the missed detections for each network are not anomalies; [8] has incorrectly identified them to be anomalies. Second, 15 anomalies in Abilene and 11 in Dante were caught by changing *Defeat*'s key definition to either use only a prefix of the source address or a prefix of the destination address. Recall that *Defeat*'s algorithm allows flexible key definitions, and consider a network scan where a single source probes a very large number of destinations. Our default key (a concatenation of source and destination IP prefixes) would map these network flows into different buckets. When *Defeat* uses only source IP prefix as the key, the entire network scan would hash to a single sketch bucket, increasing its likelihood of detection. Finally, 11 anomalies in Abilene and 8 in Dante were not caught by any of our hash functions with any key definition. We have analyzed

each one of these, and believe that *most* of these are *not* anomalies. For example, 2 detections from Abilene and 3 detections from Dante appeared to be traffic from servers to clients on different ports. Another 4 from Dante are small point-to-point flows transferring fewer than 100 small packets, and 5 from Abilene that are frequently recurring Samba over IP flows that do not appear anomalous. This leaves us with 4 and 1 true missed detections for Abilene and Dante respectively. The 4 in Abilene belong to two anomalies, a flash crowd and an alpha flow. The 1 in Dante is an alpha flow.

**IP-flow Identification.** One of *Defeat*'s advantages is that it can automatically identify IP flows that were responsible for the anomalies in a given time bin. Table 5 shows a small, randomly chosen, set of IP flows that *Defeat* identified, using the algorithm described in Section 2. In this table, we have omitted a detailed listing of the anomalous IP flows, for brevity. We have manually verified these anomalies. Unlike *Defeat*, [8] can only automatically identify the time bin during which an anomaly was detected, and the OD flows containing the anomaly or anomalies (a single OD flow might contain more than one anomaly, but that method cannot infer this).

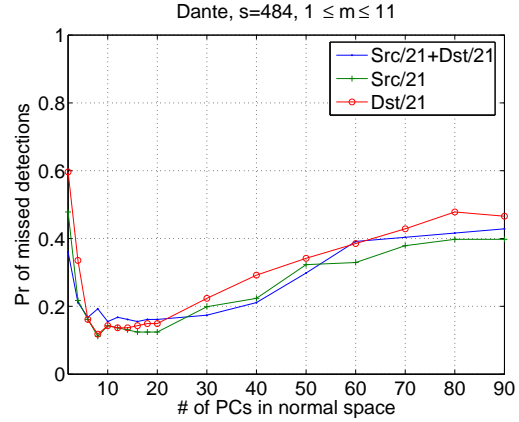
### 3.1 Exploring the Parameter Space

*Defeat* has several parameters which can affect its detection performance. We now consider the effect of varying these parameters.



**Figure 3:** The probability of missed detections as a function of the normal space size for the Abilene data set.

**Normal Space Size.** How many principal components should be used to construct the normal subspace? One way to answer this question is to examine (by plotting graphs similar to Figure 2) the eigenflows generated by each sketch corresponding to the principal components (PCs) in order, and determine that component at which the eigenflows start to differ substantially. We have observed, using this methodology, that around 10-20 PCs define the normal subspace for both data sets. Another approach is to examine how *Defeat*'s performance varies, vis-a-vis [8], with changing numbers of PCs. Figure 3 and 4 show that across different key definitions and for both data sets, *Defeat*'s missed detections are lowest when the normal subspace has 10-20 PCs. For this



**Figure 4:** The probability of missed detections as a function of the normal space size for the Dante data set.

reason, we chose 10 PCs in our evaluations; this choice is consistent with that of [8]. As an aside, notice that missed detection rates increase dramatically when fewer than 10 PCs are used; intuitively, this choice shifts higher volume normal traffic into the residual space, making it harder to detect anomalies.

In what follows, we use 10 PCs. Furthermore, we have omitted, for space reasons, the Abilene data set. However, all our observations below hold for Abilene as well.

**Sketch Size.** Figure 5 plots *Defeat* performance as a function of sketch size. Each curve represents a different sketch size, and different data points represent different votes from 11 different hash functions: the point on the top right of a curve represents a detection by at least one hash function (*union*) and the point on the bottom left represents detections by all hash functions.

Large sketch sizes decrease the missed detection rates and increase the additional detection rates. Since, as we show above, the additional detections are likely to be true anomalies, larger sketch sizes perform better than smaller ones. For our data sets, sketch size 1024 is a good choice since the probability of detections (1 - missed detection rate) is greater than 90% for both data sets. Our choice of a sketch size of 484 in our performance comparison was dictated by the need to make an even comparison with [8]; however, this figure shows that *Defeat* can perform better (detect more anomalies, with fewer false alarms) with a larger sketch size.

**Number of Hash Functions.** How many hash functions are sufficient for high detection rates? As Figure 6 shows, there exists a knee after which increasing of the number of hash functions does not improve detection rate. For most sketch sizes, this knee is the same, at 5 or 6 hash functions. While this needs more extensive data analysis, it is encouraging to note that a small number of hash functions suffices for accurate anomaly detection.

**Voting Schemes.** Figure 7 plots the detection rates against the false alarm rates for different numbers of votes. This figure uses as the definition of true anomalies those found in our detailed manual inspection, which yields a more accurate (and higher) detection probability and (lower) false

| Time | Src/21        | Dst/21        | # of flows | Anomaly type |
|------|---------------|---------------|------------|--------------|
| 206  | 137.138.248.0 | 192.101.160.0 | 495        | peer-to-peer |
| 408  | 141.85.248.0  | 64.247.56.0   | 1          | alpha-flow   |
| 417  | 157.181.192.0 | 209.200.152.0 | 3548       | dos          |
| 1049 | 202.99.240.0  | 193.2.184.0   | 2678       | ddos         |
| 1578 | 140.113.144.0 | 192.167.144.0 | 3102       | worm         |
| 1664 | 152.2.64.0    | 131.111.0.0   | 9442       | portscan     |

Table 5: Examples of anomalous flows identified by Defeat

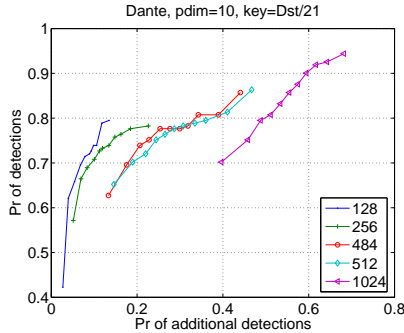


Figure 5: The probability of missed detections vs. the probability of additional detections for the Dante data set.

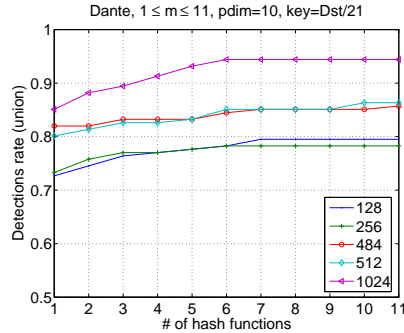


Figure 6: The detection rate as function of hash function numbers for various sketch sizes for the Dante data set.

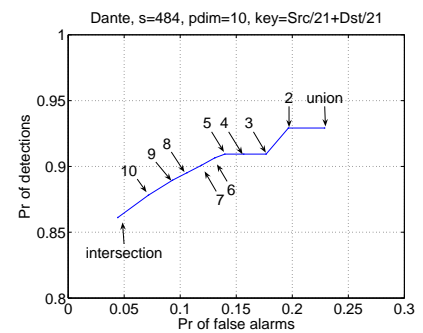


Figure 7: The detection rates vs. false alarms rates obtained with different number of votes for the Dante data set.

alarm probability than using the set of anomalies reported in [8]. Notice that intersection alone can capture more than 85% of the anomalies. To improve detection rates, *Defeat* can reduce the required number of votes at the expense of increased false alarm rates. A good rule of thumb seems to be that requiring  $m - 2$  or more votes (where  $m$  is the number of hash functions). provides good detection rates with low false alarms.

**Key Definitions** As we have discussed before, *Defeat* allows different key definitions, and different keys can unearth different anomalies (Table 4). This is also evident in Figure 3, but not in Figure 4. We have not examined the impact of key definitions more closely, and leave that to future work.

## 4. CONCLUSION

The *Defeat* algorithm presented in this paper uses multiple random traffic projections to robustly detect anomalies. In a week-long trace from the Dante backbone, *Defeat* detects nearly 200 more anomalies than prior work while missing only one. It is, in addition, able to automatically infer the IP flows responsible for an anomaly, a feature missing in any previously published work. *Defeat* brings on-line anomaly detection and identification within the realm of feasibility, which forms our future work.

## 5. REFERENCES

- [1] P. Barford, J. Kline, D. Plonka, and A. Ron. A signal analysis of network traffic anomalies. In *ACM Internet Measurement Workshop*, Nov. 2002.
- [2] C. Estan and G. Varghese. New directions in traffic measurement and accounting. In *Proceedings of ACM SIGCOMM*, Pittsburgh, PA, August 2002.
- [3] F. Feather, D. Siewiorek, and R. Maxion. Fault detection in an ethernet network using anomaly signature matching. In *Proceedings of ACM SIGCOMM*, 1993.
- [4] C. Hood and C. Ji. Proactive network fault detection. In *Proceedings of IEEE Infocom*, Apr. 1997.
- [5] I. Katzela and M. Schwartz. Schemes for fault identification in communication networks. *IEEE/ACM Transactions on Networking*, 3(6), Dec. 1995.
- [6] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen. Sketch-based change detection: Methods, evaluation, and applications. In *Proceedings of ACM Internet Measurement Conference*, Oct. 2003.
- [7] A. Lakhina, M. Crovella, and C. Diot. Diagnosing network-wide traffic anomalies. In *Proceedings of ACM SIGCOMM*, Aug. 2004.
- [8] A. Lakhina, M. Crovella, and C. Diot. Mining anomalies using traffic feature distributions. In *Proceedings of ACM SIGCOMM*, Aug. 2005.
- [9] A. Soule, K. Salamatian, and N. Taft. Combining filtering and statistical methods for anomaly detection. In *ACM Internet Measurement Conference*, Oct. 2005.
- [10] M. Thorup and Y. Zhang. Tabulation based 4-universal hashing with applications to second moment estimation. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2004.
- [11] M. Thottan and C. Ji. Anomaly detection in IP networks. *IEEE Transactions in Signal Processing*, 51(8), Aug. 2003.
- [12] Y. Zhang et al. Online identification of hierarchical heavy hitters: Algorithms, evaluation, and application. In *Proceedings of ACM Internet Measurement Conference*, Oct. 2004.