Efficient Mining of the Multidimensional Traffic Cluster Hierarchy for Digesting, Visualization, and Anomaly Identification[*]

Jisheng Wang, David J. Miller, and George Kesidis

Department of E.E., Penn State University

e-mail:jzw128@psu.edu, millerdj@ee.psu.edu, kesidis@engr.psu.edu

Tel: (814) 865-6510 Fax: (814) 865-7065

Abstract

Mining traffic traces to identify the dominant flows sent over a given link, over a specified time interval, is a valuable capability with applications to traffic auditing, simulation, and visualization of unexpected phenomena. Recently, Estan et al. advanced a comprehensive data mining structure tailored for networking data – a parsimonious, multidimensional flow hierarchy, along with an algorithm for its construction. While they primarily targeted off-line auditing, interactive visualization of current traffic or of network simulations in progress will require real-time data mining. We suggest several improvements to Estan et al.'s algorithm that substantially reduce the computational complexity of multidimensional flow mining. We also propose computational and memory-efficient approaches for unidimensional clustering of the IP address spaces. For baseline implementations, evaluated on the New Zealand (NZIX) trace data, our method reduced CPU execution times of the Estan el al. method by a factor of more than eight. We also demonstrate the usefulness of our approach for anomaly and attack identification, based on traces from the Slammer and Code Red worms and the MIT Lincoln Labs DDoS data.

Index Terms:

Network anomaly detection, Frequent item set mining, Data digesting, Hierarchical clustering, Data visualization.

[*] The work was supported by DHS/NSF grant 0335241 (EMIST/DETER).

1. Introduction

Identification of frequent item sets in databases [1] is a general-purpose data mining capability with applications to association rule mining for building expert systems, data warehousing [6], as well as general knowledge discovery. Item set mining is related to unsupervised learning, e.g. clustering [7], but is better suited to handling discrete-valued (categorical) data, for which it is difficult to find an appropriate clustering distortion measure. Recently, frequent item set mining has found applications in characterizing the content in network traffic traces [19],[9],[5] – we refer to this as *flow mining*.

While there is much prior work in measurement and content characterization of Internet traffic, including sampling techniques e.g. [8], sketches [2], and flow histogram estimation [15], flow mining offers some unique capabilities – mining traffic to identify the dominant ("heavy hitter") flows [9] and/or the *deviant* flows [5] sent over a given link, over a specified time interval, has a variety of applications in network management, simulation, and security. First, the identified dominant flows/clusters may form a concise digest, for consumption by a network administrator, to alert him/her to current patterns of traffic usage. Mined flows may pinpoint that a large amount of traffic was sent via a rarely used port number (trademark of a particular worm [3]). Flow mining could also identify an unusually large amount of traffic sent to a single IP address, indicative of a flash crowd or a DDoS attack. Second, there are applications in network security (attack-defense) simulations. For captured real traces, to be used as background traffic in security experiments, the mined flows provide a salient characterization – almost an *annotation* – of the trace's main content. Moreover, if the trace contains latent *unknown attacks*, these may be ferreted out by flow mining. Also, the identified flows can form the basis for a stochastic traffic model, to be used for (realistic) synthetic traffic generation over the experimental topology[1]. Perhaps most prominently, flow mining can be used to greatly extend and focus the powers of data visualization – tools for visualization are only as good as the data with which they are supplied. In many experimental contexts, the salient features will be unknown *a priori*. This uncertainty may stem from a lack of knowledge of the data, e.g., real background or background plus attack traffic without any annotation. It will also come from uncertainty about which traffic features and which parts of the network are impacted by a particular attack and from uncertainty about the effects of a deployed defense (e.g., auto-immune effects). Flow mining may automatically capture these phenomena which, coupled with visualization tools, allows

[1] To achieve a realistic simulation, this may require determination not only of the definitions of the significant flows, but also of the temporal duration of these flows as well as their statistical character (e.g. bursty versus persistent flows).

an experimenter to ignore vast traffic quantities while scrutinizing the most meaningful content.

Finally, as we will demonstrate, flow mining can also be a core component of an anomaly detection system, automatically identifying candidate traffic subsets – the mined flows – that are suspicious. One can then (separately) run anomaly detectors on each suspicious cluster. Since the mined flows are narrowly specified by restricted value ranges along *multiple* attribute dimensions, this allows detailed identification of anomalous flows (and, hence, possibly even of the individual packets involved in an attack). This is in contrast both to methods which require user specification of the flows to evaluate as possible anomalies and to methods which perform detection on *all* the traffic, examining fixed, single attribute fields (e.g. just looking at the range of destination IPs [11]). This latter approach may identify that an attack is present but will not in general give a precise specification of the attacking flow[2].

Previous network mining techniques can be dichotomized in several ways. One is whether *exact* frequent item set mining (flow counting) is performed [9], or only *approximate* flow counting [5],[19]. The method in [5], e.g., uses combinatorial group testing to get probabilistic guarantees on flow significance. A second categorization is whether one is seeking *dominant* (large) flows [9] or *deviant* ones [5], i.e. flows which, measured over time or space, deviate from a norm or expected value. Note that, by this definition, a flow which does *not* occur (has size zero) may be deviantly significant. However, techniques for identifying dominant flows could possibly be modified to identify deviant flows. Finally, for our purposes the most important distinction is whether the technique captures the hierarchical nature of attributes such as the source and destination IP addresses, both in defining and *identifying* prominent flows. [9] and [6] do represent hierarchical attributes while [5],[19], and [26] do not. Hierarchical representation has a long history, with hierarchical clustering and dendrograms, tree-structured vector quantization, quadtrees for images, and decision trees for classification and regression. Generally there are two motivations for using a hierarchy. First, nodes at different levels capture data at different "scales" – one may be interested both in coarse, aggregate data summaries (at higher levels), as well as in finer descriptions (at lower levels). In a networking context, hierarchical IP representation allows flow definitions based on IP address *sectors* – this may be important e.g. for identifying a DDoS attack by capturing the attacking sources in a single cluster, or for capturing a cluster that represents some collaborative Internet activity (one group of IP addresses in closed communication, or in closed communication with a

[2] The "fixed" approach will precisely specify anomalies if the anomaly is solely defined by the fixed attribute being examined, e.g. if an attack uses a rarely seen port number, all attack packets can be identified just by looking at the port value.

second group). Second, one may only care about the description at the finest scale (at the leaves). Even so, building a hierarchical structure, root-to-leaf, may be the most efficient way of *computing* the leaf layer description [9].

In *frequent item set mining*, hierarchies have played both roles. A flow is said to be "frequent"/"significant" if its aggregate traffic, measured over a given time interval, is greater than a threshold level. In [18], the main purpose of the hierarchy is to capitalize on the fact that a more specific item set (a child), based on a conjunction of attribute values, can only be significant if *all* of its (less specific) *parents* are significant. E.g., the flow specified by (source port=80, destination port=∗, protocol=TCP) is one of the parents of the flow (source port=80, destination port=1560, protocol=TCP) and has size at least as large as its child. This "significance property of the hierarchy" allows one, via a top-down, root-to-leaf process, to promptly reject many candidate item sets (those with any insignificant ancestors) and to identify the most specific frequent item sets (at the leaves) in an efficient manner. Note that, for this purpose, an item set hierarchy is constructed even if none of the items/attributes has an *individual* hierarchical nature (such as an IP address) – each level simply corresponds to a different degree of conjoined attribute specification. In [13], hierarchical mining was proposed; however, networking data was not considered. In [9], a hierarchical structure was advanced, tailored for networking data. This structure captures the hierarchical nature both of individual attributes (source and destination IPs) and collections of (hierarchical or non-hierarchical) attributes (with more specific item sets descendants of less specific ones). This multidimensional flow hierarchy defines network flows with flexible specificity. This is attractive for digesting purposes, where one may wish to view the dominant traffic at several levels of description. At the same time, [9] capitalizes on the "significance property of hierarchies" to efficiently *build* the multidimensional flow hierarchy. Thus, even if one is only interested in the most specific significant flows, it may be most efficient, computationally, to build the complete flow hierarchy, top-down, terminating in the significant flows at the leaves. While [9] did aim to optimize mining efficiency, their work was primarily proposed for off-line traffic auditing. For interactive visualization of current traffic or of network simulations in progress, as well as for anomaly detection, further improvements in efficiency are likely required. In fact, it was noted in [9] that improvements may be possible.

We suggest several modifications of [9] with the aim of greatly reducing the computations required to measure the size of flows. We also remove the potential need for sorting (and traffic sampling) in their method. We report experiments for baseline implementations on the New Zealand (NZIX) trace data [25] which verify significant gains in efficiency for our method. These improvements enhance viability of the multidimensional flow hierarchy for real-time traffic visualization and analysis. In section 2, we review [9] and describe several paradigms for improving efficiency. Section 3 reports experiments comparing efficiencies. Section 4 demonstrates on real traces that our mining tool localizes attack traffic to individual clusters. Furthermore, we propose and evaluate several criteria for automatically distinguishing "abnormal clusters" from nominal clusters. We also sketch an anomaly detection framework driven by results from the mining tool. The paper concludes with some future research directions.

2. Multidimensional, Hierarchical Flow Mining of Network Traffic

We consider flows defined by the 5-tuple (source IP, destination IP, source port, destination port, protocol). The IP addresses are hierarchical, i.e., binary prefixes of variable specificity, from 8 up to 32 bits. The protocol is treated as a flat, i.e. nonhierarchical, attribute. The ports are each represented using a very simple hierarchy, with the range first divided into low (< 1024) and high (> 1023) groups, and a flat representation for each group. Each attribute may also take on a *wild card* ($*$) value, indicating this attribute is *not* used in defining the flow. For example, (source IP=$*$, destination IP=10.0.0.1, source port=low, destination Prot=$*$, protocol=UDP) consists of all UDP traffic to 10.0.0.1 coming over low ports. The set of all possible flows, consistent with the above definitions, can be represented by a *multidimensional flow hierarchy* [9]. It is difficult to graphically depict this structure on the *full* attribute space. However, in Fig. 1, we do present an example portion of the hierarchy for the subspace ($*$,$*$,source port, $*$,protocol). We make several observations about Fig. 1. First, the root contains all the flows and the bottom (leaf) layer consists of all unique packet identifiers (as given by the attribute 2-tuple) observed during the trace, i.e. each leaf corresponds to a unique "NetFlow" table entry[3]. In-between these layers, there is redundancy in the representation, not only between layers, but also between flows at the same layer. For example, some packets contribute to the flow counts for both (lowport, $*$) and ($*$, TCP). Second, note that nearly all flows (except those one layer down from the root) have multiple parents – e.g. (80, TCP) has parents (lowport, TCP) and (80, $*$). Finally, note that if we wildcard all attributes except for one, we are considering a unidimensional sub-graph of the multidimensional hierarchy. Unidimensional hierarchies play an important role in [9] and our extension.

[3] For the illustrative example in Fig. 1, which considers a 2-D subspace, the "NetFlow table" entries consist of the 2-tuples (source port, protocol). In practice, for the full 5-D space, the table entries consist of unique 5-tuples (source IP, destination IP, source port, destination port, protocol).

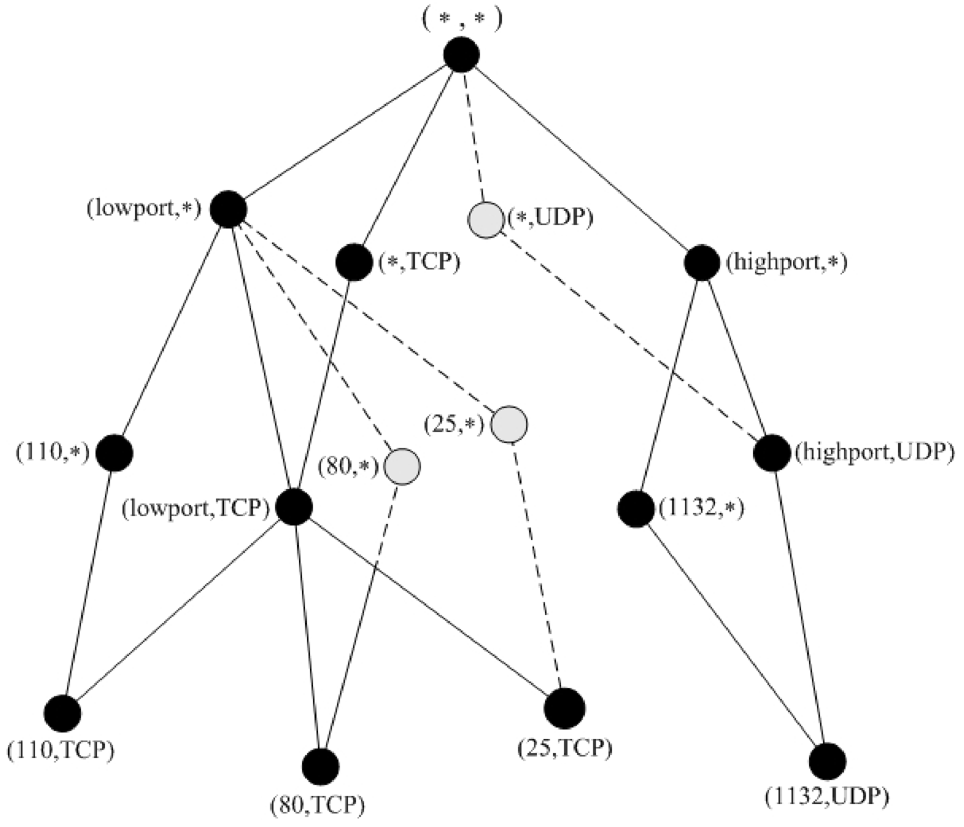

Figure 1: Part of the flow hierarchy for the 2-D subspace consisting of the source port and protocol.

The mining objective in [9] is twofold: first, identify *all* nodes in the flow hierarchy with aggregate flow count (measured over the time window) greater than a specified threshold. The count may be measured either in packets or bytes. Second, since, for a given threshold, there may be hundreds or *thousands* of such "significant" flows, the authors proposed a simple method for greatly reducing this number to achieve a concise digest. This amounts to a simple "compression" algorithm, removing those (many) flows whose sizes can be predicted sufficiently accurately, based on the sizes of the significant flows retained in the digest. We next describe the mining in [9] in more detail.

2.1 Identifying Significant Unidimensional Flows

The method starts by considering the (unidimensional) hierarchy for each attribute, identifying, for this single attribute, *all* "significant" flows. This can be easily done for the port and protocol attributes. For the IP attributes, the authors identify all significant flows in a bottom-up fashion. First, taking a single pass over the trace (or over the derived NetFlow table), the size of each leaf is computed. Next, the hierarchy is traversed from leaf up to root, aggregating the counts of all children

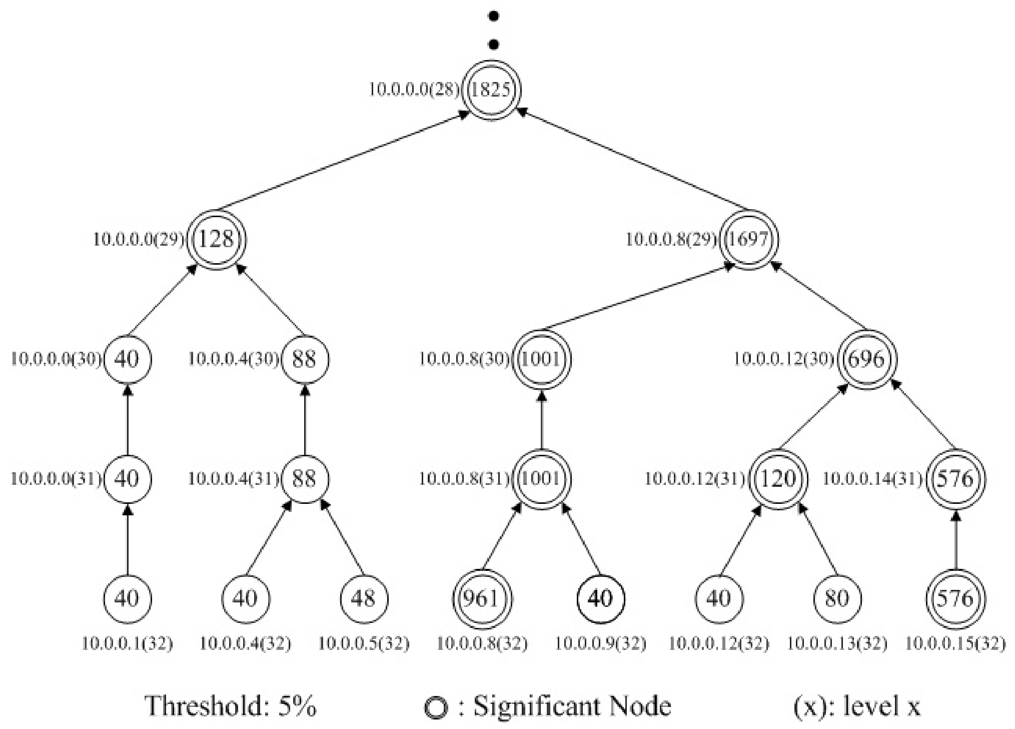

Figure 2: A portion of the 1-D IP address hierarchy, with significant nodes identified. The IP address level is shown in parentheses.

in calculating the count of their parent[4]. Once computed, a node's count is compared to the threshold to determine significance. Fig. 2 shows an example portion of the IP address hierarchy with some significant nodes. Only these nodes need be stored for later use. Since these flows (as derived from the trace or the NetFlow table) will not be initially ordered in practice, one must either use a sorting algorithm, static memory of size equal to the range of IP addresses that occur in the trace, or some type of dynamic memory allocation strategy. Since the range of IP addresses could be nearly the full (32-bit) range, static memory allocation could require (an unacceptable) 4-gigabytes of memory. Sorting amounts to ordering 32-bit unsigned integers, with complexity, for N addresses, ranging from $O(N \log N)$ to $O(N^2)$ for standard methods. For increasing time windows, the number of addresses may grow well into the millions, in which case the computation for leaf creation and sorting (and the required memory) may become significant. To reduce the number of leaves, flow sampling may be performed [10]. This will, however, reduce accuracy of the counts (and could result in missing some significant flows). We later propose an alternative to "bottom-up" unidimensional clustering which circumvents some of these problems.

[4] For unidimensional hierarchies, there is a single parent for each child.

2.2 Identifying Significant Multidimensional Flows

While unidimensional clustering has digesting value on its own, its primary role is in speeding up the (subsequent) algorithm for building the *multidimensional* hierarchy. Whereas the unidimensional hierarchies are built in a bottom-up fashion from the leaves, the multidimensional hierarchy is built *top-down*, level-by-level, starting from the root (k = 1). In going from level $k-1$ to level k, one is further specifying a single attribute value. The root node is always significant. The next level consists of (partitioned) flows along each attribute dimension, e.g., for the source IP, (‘0∗’, ∗, ∗, ∗, ∗) and (‘1∗’, ∗, ∗, ∗, ∗), for the source port, (∗,∗,lowport, ∗, ∗) and (∗,∗,highport, ∗, ∗), and for the protocol, (∗, ∗, ∗, ∗, protocol id). Each flow at this second level is checked for significance. This is done by matching the flow’s definition against the definition of each flow in the NetFlow table and, if a match occurs, adding to the current accumulated flow count. Upon completion, if the flow count is above the threshold, the flow is saved as “significant”. For checking each flow, one thus requires, for a NetFlow table with L entries, *L match operations*, as well as the additions. It is this matching operation which will be the focus of our algorithm improvements. For each subsequent level, k = 3,4,.,leaf-depth, [9] identifies significant nodes in the hierarchy via the algorithm in Table 1. Some

```
Begin{multidimensional clustering}
1:  for $k = 3$ to maxdepth
2:      for each significant node at level $k-1$:
3:          1. Identify all of the node’s children (at level k).
4:          2. for each child, if its flow size has not already been checked:
5:              i. Check whether all the 1-D ancestors of the flow are significant.
6:              ii. If i. is true, check whether all parents of the flow are significant.
7:              iii. If i. and ii. are true, measure the flow size using the NetFlow table, as previously
8:                  discussed. If the flow is above the threshold, save it (and its size) as a “significant”
9:                  node in the hierarchy.
10:         end for
11:     end for
12: end for
End{multidimensional clustering}
```

Table 1: The algorithm pseudocode for multidimensional clustering.

explanation is in order. First, the algorithm tests two necessary conditions for flow "significance" before resorting to actually measuring the size of a flow. These low-complexity tests reduce the number of (computationally heavy) flow count passes taken over the NetFlow table. The first test checks significance of all unidimensional ancestors. For example, the flow (10.0.0.1, ∗,80, ∗,TCP) has the 1-D ancestors (10.0.0.1, ∗, ∗, ∗, ∗), (∗, ∗,80, ∗, ∗), and (∗, ∗, ∗, ∗, TCP). Note that whether these flows are significant was previously determined by unidimensional clustering. The second test capitalizes on the "significance property of the hierarchy" to reject flows that have any insignificant parents. Also, as stipulated in step 2., the operations i., ii., and iii. are only done if the flow has not already been checked. This accounts for the fact that, except at level 2, a child has *multiple* parents. Redundant checking of a child flow is thus avoided. In [9] the authors also briefly indicated, albeit without any details, a "batching" procedure wherein, for a single pass over the NetFlow table, the sizes of *multiple* flows (each satisfying i. and ii.) are calculated – this can potentially reduce flow checking computations. We will discuss a possible batching scheme in the results section. Finally, after identifying all significant nodes, the authors proposed a node "compression" algorithm. First, all significant nodes without significant descendants (i.e., the most specific subset of significant flows) are retained. Then, traveling up, level-by-level towards the root, significant nodes are *only* retained if their sizes are not well-predicted, based on aggregation of their retained descendants. The remaining "compressed hierarchy" is generally much smaller than the initial hierarchy of significant nodes. This "compressed hierarchy" can predict *all* flows to within a specified tolerance.

2.3 Improving the Efficiency of Multidimensional Flow Mining

The algorithm [9] in the last section consists of *unidimensional* and *multidimensional* steps. The majority of the execution is spent on multidimensional clustering[5]; moreover, the bulk of this execution is the flow counting in step 2.iii, i.e., matching a flow's definition against the NetFlow entries and aggregating byte and packet counts. Moreover, as seen in our simulations, for increasing trace lengths, step 2.iii accounts for an *increasing* percentage of the total execution time. Thus, how efficiently the flow counting is performed may critically determine whether the mining process can be implemented in real-time, for visualization and anomaly detection. In [9], it was stated that "our algorithm examines all clusters that may be above the threshold; for each such cluster, the algorithm examines all *n* flows [in the NetFlow table], and adds up the ones that match". Suppose there are *N*

[5] However, as will be discussed further, unidimensional clustering as performed in [9] can also be time-consuming unless static memory is allocated for the IP address spaces.

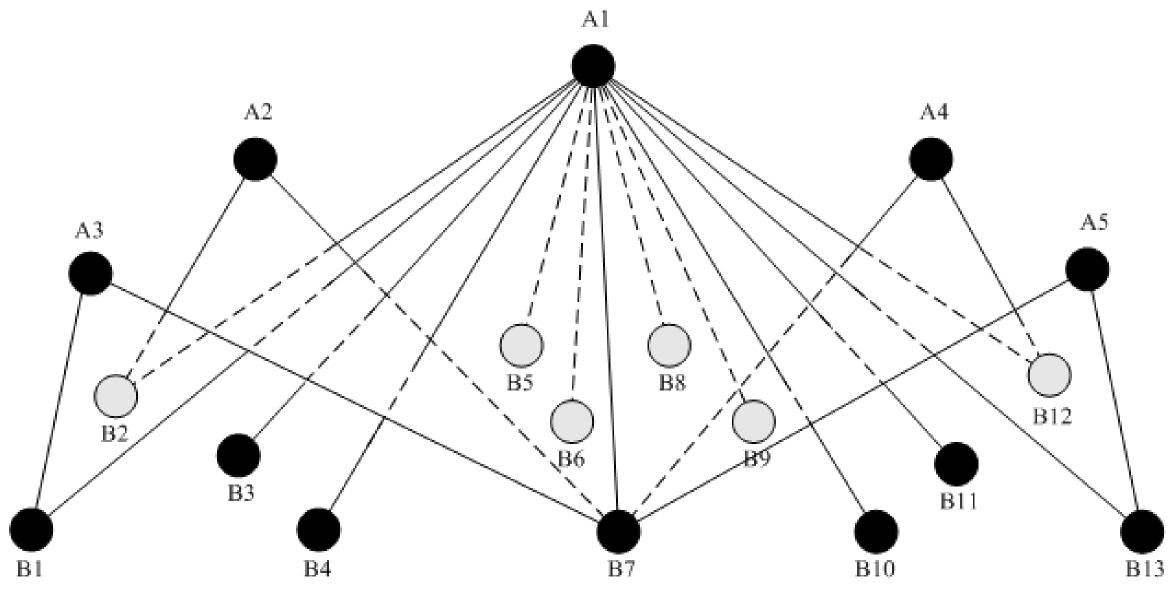

Figure 3: Part of the flow hierarchy for a 5-D multidimensional tree.

candidate clusters in the multidimensional tree which satisfy the first two tests, i.e., for which it is necessary to do flow counting. For a NetFlow table with L entries, the total flow counting time is $O(NL)$, with L (and possibly N) growing with the duration of the trace. In this section, several strategies are presented to greatly reduce flow counting time. Fig. 3 shows an example portion of two adjacent levels of the multidimensional tree. Clusters A1 through A5 are distributed on one (parent) level and clusters B1 through B13 are on the next (child) level. Suppose that A1-A5 are *all* significant and that we are currently working to identify the significant children of A1. B1-B13 are all children of A1. B1 and B2 are the children (siblings) for the source IP, B3 and B4 are siblings for the destination IP, B5, B6 and B7 are for the source port, B8, B9 and B10 are for the destination port, and B11, B12, and B13 are for the protocol[6]. We use B7 to illustrate our flow counting strategies. Suppose we have finished the first two tests (2.i and 2.ii) for B7, as described in Section 2, and have found that all five 1-D ancestors and all five parents of B7 are significant. It is then necessary to perform step 2.iii.

Flow Subset Paradigm:

In the following, we use {X} to denote the subset of flows in the NetFlow table which match flow definition *X*. As described in [9], one needs to go through all *L* flows in the NetFlow table to

[6] There are only two children for both IPs, while there may be more for the other three dimensions.

determine which ones match B7's definition, aggregating their byte and packet counts to precisely measure B7. We greatly reduce this operation by capitalizing on the flow nesting inherent in the hierarchy. As shown in Fig. 3, if a certain flow in the NetFlow table matches B7's definition, it must also match A1, as well as B7's other four parents, A2-A5, i.e. a child flow is a subset of each of its parent's flows. Thus, suppose, while doing flow counting at the A level, that we *saved* the flow subsets for the significant nodes at this (parent) level. Then, to perform flow counting for a node at the child level, we can perform the matching operation *using the flow subset table of the node's parent*, rather than the entire NetFlow table. For example, to measure the size of B7, we can simply identify the flows in {A1} which match B7's definition. |{A1}| is generally *much* smaller than *L*. In fact, considering the top-down process for building the multidimensional tree level-by-level, the size of a flow subset in general decreases *exponentially* with increasing level. Thus, applying the strategy of matching against the parent flow subset table, rather than against the original NetFlow table, should yield large reductions in flow matching operations. We report simulation results in the next section. This "Flow Subset Strategy" does entail increased temporary storage – to perform counting at level *k*, we require temporary storage of all significant flow subsets at level $k-1$. Moreover, we must also temporarily save all identified significant flow subsets at level *k* (for subsequent use in flow counting at level $k+1$). Thus, at any given time, we must reserve storage for the significant flow subsets at two adjacent tree levels. We allocate memory to handle the *maximum* (worst case) storage requirement, over all tree levels. Alternatively, a dynamic memory strategy could be employed. Regardless, in practice the memory requirements of this scheme are manageable, even for lengthy traces. Results are reported in the next section.

Minimal Parent Strategy:

In flow counting for B7, we focused on the parent A1. However, B7 has, in general, *five* parents, and if step 2.iii is required, it means all of these parents are significant. Moreover, in performing the Flow Subset Paradigm, all five parent flow subsets will have been saved. Thus, in flow counting for B7, we need not restrict ourselves to using A1 as the substitute for the NetFlow table – we could consider any of the parents, since $\{\mathrm{B7}\} \subseteq \{\mathrm{A1}\},\{\mathrm{A2}\},\{\mathrm{A3}\},\{\mathrm{A4}\},\{\mathrm{A5}\}$. More precisely, $\{\mathrm{B7}\} = \bigcap_{i=1}^{5} \{Ai\}$.

While *explicitly* finding the set of intersection (by subset comparisons) does not appear to be efficient, we propose, instead, to use the parent with *minimum* subset size, i.e., to replace the NetFlow table by $\arg\min_{\{A_i\}} |\{A_i\}|$. This strategy, dubbed by us the Minimal Parent Strategy, significantly reduces flow counting operations, relative to always using A1.

Complement Strategy:

Consider partitioning the candidate nodes at a given level into distinct *sibling* groups – the group size will be two an IP dimension, and possibly (even much) greater than two for other attributes. The clusters comprising any such group have flow subsets that are mutually exclusive and collectively exhaustive of the parent subset. For example, B5, B6, and B7 are siblings and children of A1 for the source port. Accordingly, when we flow count B6 (after first flow counting B5), we can simply exclude all of B5's flows. Likewise, after excluding the flows of both B5 and B6, the remaining flows must be B7's. Thus, no matching operation is required for B7. We refer to this as the Complement Strategy.

Top-Down Unidimensional Clustering for IP Dimensions

The unidimensional clustering for the two IP dimensions is performed bottom-up in [9], i.e., these binary trees are built up from the leaves, which correspond to all distinct IP addresses in the NetFlow table. There are problems inevitably encountered during this bottom-up process: the computational complexity for creating the leaves and sorting them in an increasing order. In our results, we have found that leaf creation in particular is heavy computationally. These operations can in principle be avoided if the two IP address spaces are mapped to static memory. Unfortunately, because 2^{32} addresses may be needed for each IP space, this static memory may (impractically) require more than 4 GB of memory. To have an efficient method both computationally and memory-wise, we propose to use *top-down* unidimensional clustering, from the root to the leaves, based on a specialization of our top-down *multidimensional* clustering. Proceeding top-down, we remove the need to first create and sort leaves – only leaves which satisfy step 2.ii[7] are created. Moreover, the "Flow Subset" and "Complement" strategies greatly reduce flow checking. Because each child has only one parent, the "Minimum Parent" strategy is not needed in the unidimensional case. For the "Flow Subset" strategy, because all flow subsets on the same level are mutually exclusive, the aggregate size of all subsets at any level is at most the NetFlow table size; also, this aggregate size decreases as we top-down create the tree, level by level. Moreover, the "Complement" and "Flow Subset" strategies together guarantee that, at each level, this method only needs to check each input flow at most once. Thus, an upper bound on the number of flows checked is $O(dL)$, where d is the maximal depth of the IP hierarchy. We have verified in experiments (see section 3) that the processing time for this top-down uniclustering method is almost as small as for bottom-up clustering using static memory for IPs, is

[7] Step 2.i is not needed in the unidimensional case.

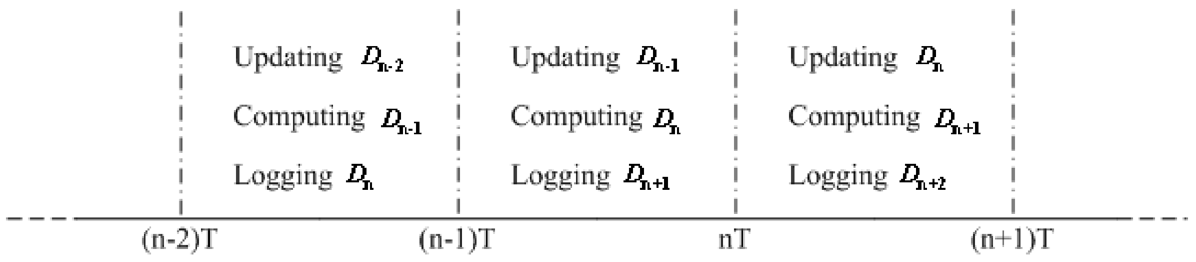

Figure 4: Illustration of online mining implementation.

less than 0.5% of the total flow mining time, and is much less than bottom-up clustering with creation and sorting of leaves. Finally, as noted in section 2, traffic sampling may be required to make leaf creation and sorting practical when the set of distinct IPs is large. Since our approach does not require these operations, it may avoid the need for sampling.

2.4 Implementation Considerations

There are several implementation contexts for on-line mining, ranging from 1) on line card in concert with the network processor or the local host processor to 2) off line card using the host processor of the entire router. The proper context depends on the costs of a variety of factors including sampling rate, available memory for logging packet information and performing computation, and available computing power. Given a choice of context, let *T* represent the amount of execution time required to perform a complete digest on a group of *L* logged packets (with unique NetFlow table entries)[8]. Also assume there is memory for logging more than *2L* packets. For each integer *n*, let the digest D_n represent the traffic that occurred over $[(n-2)T,(n-1)T]$. Fig. 4 shows the operational cycle for on-line digesting. Since the number of packets may be *much* less than *L*, the execution time may be *much* less than *T* seconds and the digest D_n could be treated as the current digest as soon as flow mining terminates, some time within $[(n-1)T,nT]$. However, for clarity's sake, suppose D_n is treated as the current, "operating" digest starting only at *nT*. During $[nT,(n+1)T]$, D_n can be sequentially modified to account for the packets that arrived (and were logged) over $[(n-1)T,nT]$. One possibility is simply to update the "bin counts" for the flows in the existing digest D_n. Since updating counts is much less complex than performing the mining operation, this type of updating is quite feasible. Another possibility is to perform more complex split/merge operations on the hierarchy comprising D_n, in response to the packets that occur within $[(n-1)T,nT]$.

[8] We assume that the number *L* gives a figure of merit for the required execution time.

3. Experiments Comparing Computational Efficiency

We tested both our new method and [9] on the New Zealand (NZIX) trace data [25] for 30-minute, 1-hour, 2-hour, and 3-hour traces, with the significance threshold set at 5% of the total traffic byte count. Since we did not have source code for [9], we created our own implementation. Both methods were coded in C, without substantial optimization, and run on a Windows XP, at 3 GHz with 50% usage. For [9], we implemented a batching strategy, wherein all children of a given parent are checked during a single NetFlow table pass – if a flow fails to match the parent, one need not check if it matches any children. For both our method and [9], we initially performed top-down unidimensional clustering of IP addresses. Using the top-down approach, the complexity of unidimensional clustering for the IPs is very low, i.e. multidimensional clustering takes nearly all execution time, for both methods. This allows us to focus comparison between methods to a comparison of multidimensional clustering approaches. However, note that we will also shortly explore the practical advantages of our top-down uniclustering of IPs, compared with bottom-up uniclustering.

We found that, for both multidimensional clustering methods, step 2.iii (Table 1), flow counting, takes more than 95% of the total time (for all trace lengths). In Table 2, we compare in terms of the total number of flows checked in step 2.iii (the "Flow" row) and the total execution time. In checking whether a given NetFlow entry matches the current flow definition, we match *attribute by attribute* (considering the IPs last) and terminate as soon as a match fails. Thus, we expect the total number of *attributes* checked, rather than the number of flows, better reflects the required processing time. Accordingly, we have included a row in Table 2 (labeled "Attribute"), with the total number of attributes checked. From the "Attribute Ratio" and "Time Ratio" rows, our clustering method has reduced the number of attribute matches by a factor of 9.40 (averaged over all four trace lengths), and the total time by an average factor of 8.21, compared with [9]. For the 1-hour New Zealand trace, the execution is less than 13 seconds for our method. We believe this new method will be useful in real-time networking applications.

We also calculated a matching "efficiency" measure, the ratio of the aggregate number of attribute matches to the number of flow matches. As discussed before, flow matching is performed attribute by attribute. The more attribute tests required for a flow, the closer the flow is to matching the definition. Thus, the *average* number of attributes checked per flow gives the efficiency of flow matching – 5.0 indicates perfect efficiency, i.e., one is only checking flows that match the target

Trace Data Length			30 min	1 hour	2 hours	3 hours
Matching Operation Comparison	New Method	Flow	7,633,526	18,893,753	35,438,798	69,457,759
		Attribute	35,741,232	89,856,622	167,388,369	332,764,019
		Efficiency	4.68	4.76	4.72	4.79
	Method in [9]	Flow	278,466,597	524,625,237	1,153,466,964	1,917,679,349
		Attribute	386,276,953	833,210,066	1,587,875,389	2,674,029,473
		Efficiency	1.39	1.59	1.38	1.39
	Attribute Ratio		10.81	9.27	9.49	8.04
Processing Time Comparison	New Method		5.922s	12.766s	23.344s	45.360s
	Method in [9]		48.907s	101.329s	206.422s	354.156s
	Time Ratio		8.26	7.94	8.84	7.81
Maximum Memory Req. (Byte)			1,623,233	3,860,483	7,220,682	12,836,127

Table 2: Experimental results for the New Zealand (NZIX) trace data.

definition. A much smaller number means that many candidate flows being checked are not very good candidates. The matching efficiency of our new method averaged over all trace lengths is 4.74, while the efficiency of [9] is only 1.44. These numbers further explain why it is the ratio of the total number of attributes checked, not the number of flows checked, which gives a good indication of the time reduction factor for our method, relative to [9]. In particular, from Table 2 the ratio of the number of flows counted is as high as *thirty*, but the speedup factor of our method is only between 7.8-9. – [9] is checking many more flows but fewer attributes per flow. Table 3 indicates the percentage reduction in time associated with each of our efficiency strategies. Much of the speedup is attributable to our “Flow Subset” strategy, with gains also coming from the “Minimum Parent” strategy. In Table 2, the maximum memory required for saving the flow subsets for a 3-hour trace was less than 13 MB, a value which should be quite acceptable on most computing platforms.

Comparison of the execution times for top-down unidimensional clustering (UC) and bottom-up UC is shown in Table 4 for the same NZIX trace as above. In the NZIX trace, the IP addresses (presumably anonymized) occur in increasing order in the trace, which obviates the need for a sorting operation. However, in practice this will not be the case. Moreover, execution time for node creation

Trace Data Length			30 min	1 hour	2 hours	3 hours
Attribute Matching Operation Comparison	Method in [9]		386,276,953	833,210,066	1,587,875,389	2,674,029,473
	Number of Attribute Matching	FS	41,732,049	101,882,780	191,784,251	387,489,430
		FS&MP	37,219,435	92,632,609	173,229,165	342,756,406
		All Three	35,741,232	89,856,622	167,388,369	332,764,019
	Percentage Reduction	FS	825.61%	717.81%	727.95%	590.09%
		MP	12.12%	9.99%	10.71%	13.05%
		Comp.	4.14%	3.09%	3.49%	3.00%

Table 3: Complexity reduction associated with each of the three new strategies.

Trace Data Length		30 min	1 hour	2 hours	3 hours
Top-Down UC		0.031s	0.062s	0.094s	0.140s
Bottom-Up UC in [9]	Averaged Time	0.361s	1.105s	3.695s	7.169s
	Standard Deviation	0.0047	0.0072	0.0077	0.0094

Table 4: Computation time comparison of top-down and bottom-up unidimensional clustering.

and sorting in bottom-up UC will depend on the order. Accordingly, we randomly ordered the input NetFlow table and measured the average execution time and the standard deviation for bottom-up UC, based on 10 different randomly ordered traces[9]. Because computer memory is always affordable for the port and protocol dimensions, we used the same static memory allocation method for the source port, destination port, and protocol dimensions for both methods. For bottom-up UC, we used *quicksort* to sort all IP leaves before clustering bottom-up for both source and destination IPs. As shown in Table 4, because it avoids creating and sorting leaves[10], the new memory-affordable top-down UC is much more computationally efficient than the bottom-up method for the IPs.

[9] Top-down unidimensional clustering execution time is independent of the order of the input NetFlow table.

[10] Creating unidimensional leaves from the input NetFlow table takes most of the computation time of the bottom-up method.

4. Experiments with Attack Traffic and Anomaly Identification

In this section we demonstrate the potential of our mining for localizing attack traffic to individual clusters. We also propose a novel criterion useful for discriminating attack from nominal clusters. We refer to this objective as *anomaly/attack identification*. If the unit time interval for mining, *T*, is sufficiently large that the attack is wholly or mostly contained in a single report, then anomaly identification will discern attacks that are either mature or already completed. On the other hand, if *T* is made sufficiently small, mining may capture the attack at an early stage. In this case, there are two possibilities. One is simply to treat identified anomalous clusters *as attack detections*, i.e. to consider anomaly identification as the detection mechanism. Alternatively, one can treat these clusters as "suspicious" and run anomaly detectors, e.g. [14], (possibly looking at several attributes) on the traffic subset that matches the cluster's definition. Both approaches will narrowly localize attacks to multidimensional clusters. The latter two-stage approach allows the use of sequential detection theory to make decisions at any time (not just at digest intervals) and, in principle, to make decisions consistent with a given (false alarm, missed detection) tradeoff. This two-stage process, applied to individual clusters, should localize attacks much better than systems which focus only on a single attribute. Moreover, the complexity will be much less than systems which do localize attacks, but only by running separate detectors on *many* multidimensional flows (e.g. every flow specified by a unique source or destination IP). Our approach concentrates detection efforts on a small number of significant, suspicious, multidimensional flows. While our results will focus on attack identification, the suggested two-stage process will be considered in future work.

Data Sets

One of the main barriers to advances in attack detection is the dearth of publicly available annotated traces from the operational Internet. We have considered the primary traces of which we are aware: the DDoS data from the DARPA study [17], the Sapphire/Slammer worm traces [23], and the Code-Red version 2 traces [22].

Criterion for Anomaly Identification

Many criteria have been proposed in the attack detection literature, often capitalizing on known signatures. These include, e.g., monitoring (unusual) port numbers (worms), the distribution of destination IPs (worms), the distribution of source IPs (flash crowds, DDoS), and the atypical size of a flow. In [9], a size criterion, dubbed "unexpectedness", was proposed based on the discrepancy between the actual traffic volume of a cluster and the volume predicted based on a model that assumes statistically independent attributes. This criterion is defined, for cluster *k*, as:

$$unexpectedness(k) = 100 * \frac{p(k)}{\prod_{i=1}^{5} P_i(k)}$$

where $p(k)$ is the fraction of traffic that falls in cluster *k* and $P_i(k)$ is the fraction that matches the i^{th} attribute value of cluster *k*.

To reliably detect a variety of exploits, multiple criteria may need to be evaluated. However, this can also complicate the system design, e.g., entailing proper choice of multiple decision thresholds, one for each (scalar) criterion. Moreover, unless each criterion is exclusively an indicator for a single type of attack, one cannot ignore the interplay between criteria[11]. For the traces considered here, with latent DDoS and worm traffic, we have found a single criterion that is quite discriminative between "nominal" and "attack" clusters. For a given cluster, *k*, we define the source (destination) IP entropy as:

$$H_k(IP) = -\sum_l P[l] \log P[l] \qquad l\text{: IP addresses in cluster } k \text{ that occurred in the current digest interval}$$

where

$$P[l] = \frac{\text{number of packets (bytes) in cluster } k \text{ with IP} = l}{\text{number of packets (bytes) in cluster } k}.$$

For packet (byte)-based reports, the probabilities $P[l]$ are computed based on packets (bytes)[12].

For DDoS attacks, within a cluster that contains all the attacking sources, we would expect the source IP entropy to be much larger than the destination IP entropy. On the other hand, for a fast scanning worm attack, "worm clusters" should have a much larger destination IP entropy than source IP entropy (a few infectives seeking many new hosts). We combine these two entropies, forming a single scalar decision statistic which we dub the *absolute IP difference in entropy* (AIDE), i.e.

$$AIDE(k) = \left| H_k(source\ IP) - H_k(destination\ IP) \right|$$

In the sequel we evaluate the discrimination power of AIDE on the three attack traces. To give a reference for comparison, we also evaluate the "unexpectedness" measure from [9].

[11] For example, how should one interpret exceeding a threshold for one criterion that is a good indicator for worm attacks but failing to exceed the threshold for a second worm criterion?

[12] In some cases, one can imagine that even if the mining report is byte-based, the entropies should be packet-based. However, we have not evaluated this criterion in this work.

Leaf and Internal Node Clusters

In the report we can distinguish two types of clusters: 1) leaves and 2) internal clusters. Leaves do not have significant descendants[13]. These are the most *specific* significant flows. Summing the sizes of all the leaves gives a good estimate of the total traffic volume. All other significant clusters in the report are internal nodes. Attack traffic will always be captured in both leaf and internal clusters. However, attacks will become diluted by nominal traffic as one moves, starting from an attack leaf, up to its internal cluster antecedents. It is possible an internal cluster may consist of pure attack traffic[14]. However, the attack is also captured by multiple leaves. Moreover, for the traces considered here, we have not found that internal nodes add substantial information. Thus, in order to simplify our results display and discussion, we will focus here on the leaves.

4.1 Attack Identification Results

1). DARPA Trace

Here there is a DDoS attack to IP 131.84.1.315 and "all packets have a spoofed, random source IP" [17]. The multidimensional report for DARPA is shown in Table 5[15]. We sorted the significant clusters in decreasing order of AIDE value and only listed the top 10 leaf clusters. The first two clusters on the list are pure DDoS traffic to destination IP 131.84.1.315. In Fig. 5, the "unexpectedness"[9] (based on the number of packets) is compared with AIDE. Circles denote DDoS clusters, while 'pluses' denote nominal clusters. The two DDoS clusters have much higher AIDE than the nominal clusters; thus, AIDE is a very good decision statistic for this trace. The DDoS clusters also do have high "unexpectedness" (2,332 % and 2,447%). However, these clusters only rank 5th and 6th for this criterion with values much smaller than the top-ranking cluster's (320,263 %).

2). Sapphire/Slammer Trace

Sapphire/Slammer, a random scan UDP worm, was "the fastest computer worm in history" [20]. The worm is identified by "the presence of 376-byte UDP packets", which appear to be "originating from seemingly random IP addresses and destined for port 1434/udp" [27],[16]. The multidimensional (byte) report is shown in Table 6. The first, third, and fourth AIDE-ranking clusters are pure worm,

[13] For example, for the unidimensional report in Fig. 2, the leaf clusters are the nodes with traffic sizes 128, 961, 120, and 576.

[14] For example, if there are several sources of a DDoS attack with traffic size above the threshold, there will be attack leaves in the mining report, as well as possibly an internal node that captures the attack packets from all the sources. This internal node would certainly be of interest for attack identification since it fully captures the attack.

[15] These DDoS packets always have small payloads, so we used the packet report (not the byte report) to detect DDoS attacks.

No	Src IP	Dst IP	SrcPt	DstPt	Pr	Byte	Packet	Perc	AIDE
1	0.0.0.0/2	131.84.1.31/32	high	high	6	0.9M	23.9k	7.1%	10.11
2	64.0.0.0/2	131.84.1.31/32	high	high	6	0.9M	23.8k	7.0%	10.10
3	128.0.0.0/1	172.16.112.0/23	80	high	6	13.5M	18.6k	5.5%	2.61
4	0.0.0.0/0	172.16.116.0/23	low	high	6	10.5M	17.0k	5.0%	2.59
5	172.16.116.0/23	0.0.0.0/0	high	low	*	1.3M	17.8k	5.3%	2.50
6	128.0.0.0/1	172.16.116.0/23	low	high	*	10.1M	18.3k	5.4%	2.48
7	172.16.112.50/32	128.0.0.0/1	*	*	6	0.8M	17.1k	5.1%	2.32
8	172.16.112.50/32	128.0.0.0/2	*	*	*	0.8M	17.2k	5.1%	2.28
9	192.0.0.0/3	172.16.112.0/22	80	high	6	11.9M	18.6k	5.5%	2.26
10	172.16.112.0/22	192.0.0.0/3	high	80	6	1.7M	17.0k	5.0%	2.24

Table 5: Multidimensional clustering report of DARPA trace.

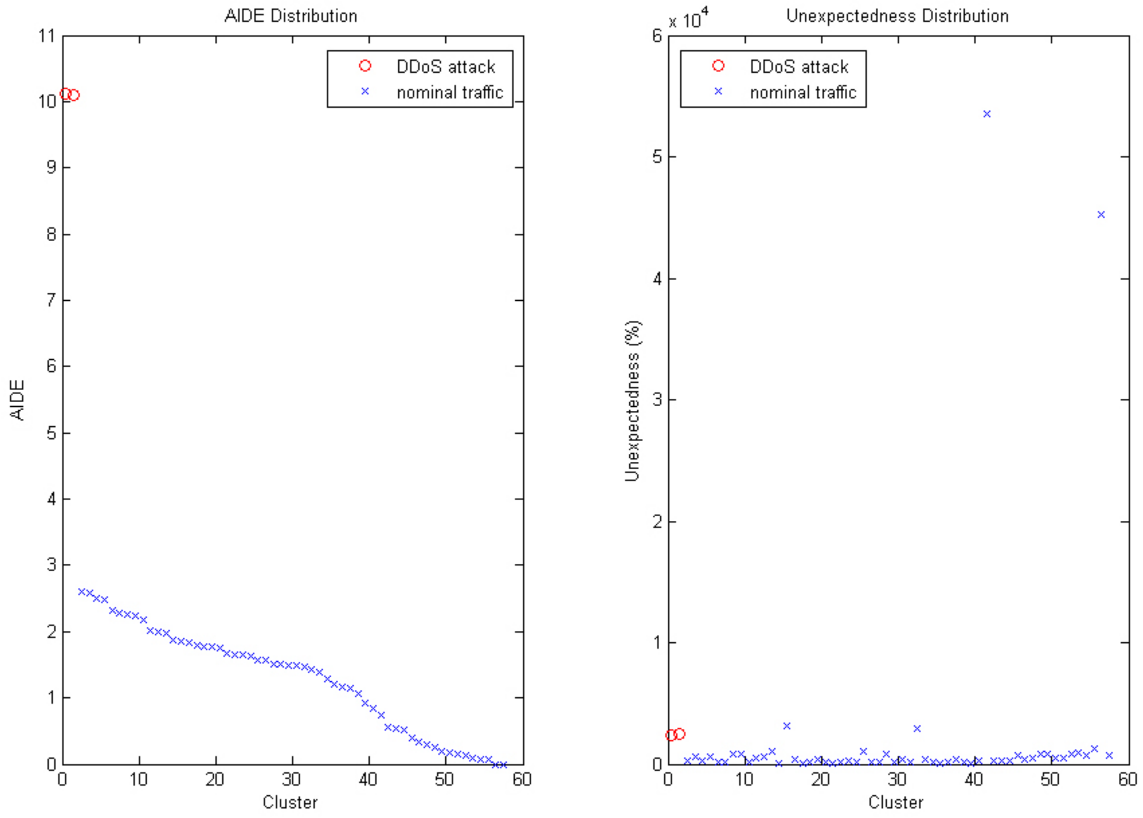

Figure 5: AIDE and unexpectedness distribution of DARPA trace.

No	Src IP	Dst IP	SrcPt	DstPt	Pr	Byte	Packet	Perc	AIDE
1	10.0.0.0/23	10.0.0.0/16	high	1434	17	11.9M	30.1k	7.0%	7.69
2	10.0.0.0/25	10.0.0.0/16	high	high	17	8.8M	37.1k	5.1%	6.81
3	10.0.0.0/21	10.0.128.0/17	high	1434	17	8.8M	22.2k	5.2%	5.98
4	10.0.0.0/21	10.0.0.0/17	high	1434	17	9.9M	25.2k	5.8%	5.84
5	10.0.0.0/28	10.0.0.0/17	high	high	*	9.6M	14.4k	5.6%	2.85
6	10.0.0.0/23	10.0.0.0/17	high	high	17	11.2M	70.7k	6.6%	2.13
7	10.0.0.0/27	10.0.0.0/18	high	high	*	8.8M	16.0k	5.2%	1.32
8	10.0.0.0/16	10.0.8.0/21	high	high	6	10.5M	10.8k	6.2%	0.69
9	10.0.0.0/21	10.0.0.192/29	high	high	6	8.5M	5.9k	5.0%	0.21
10	10.0.0.32/27	10.0.0.32/27	high	high	6	11.5M	30.5k	6.7%	0.21

Table 6: Multidimensional clustering report of Slammer worm trace.

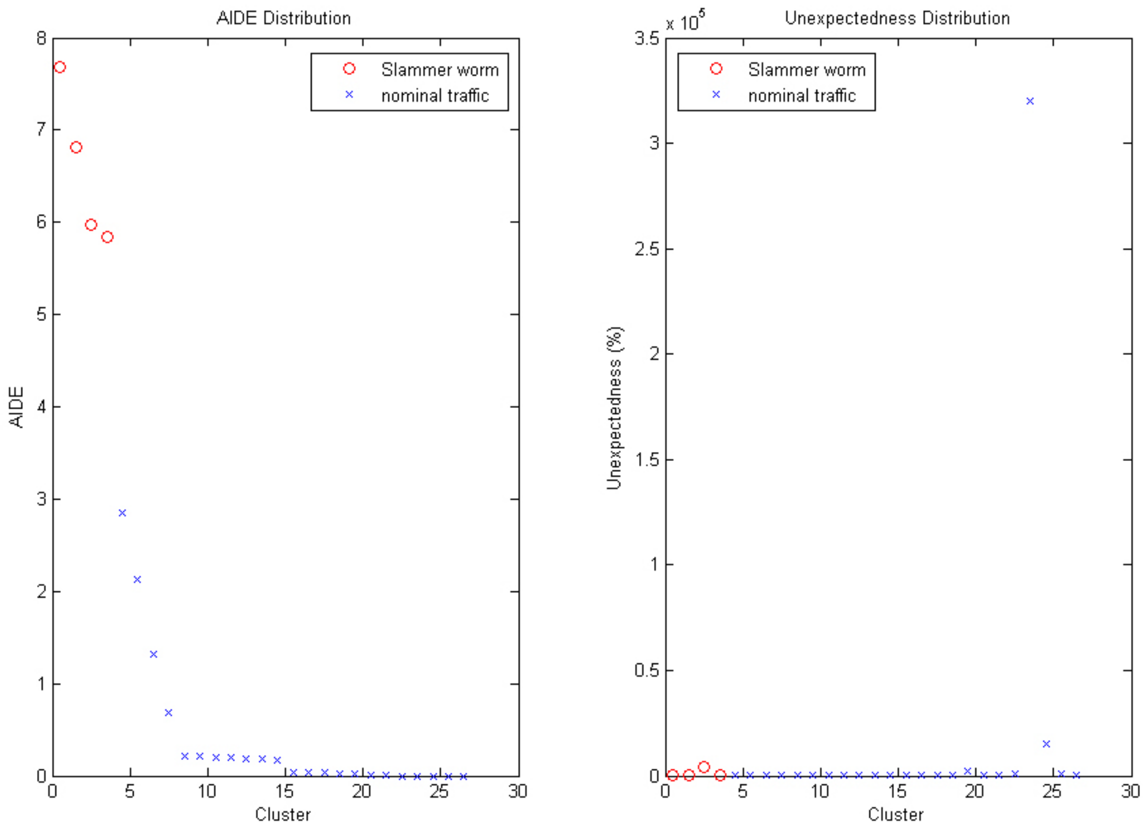

Figure 6: AIDE and unexpectedness distribution of Slammer worm trace.

No	Src IP	Dst IP	SrcPt	DstPt	Pr	Byte	Packet	Perc	AIDE	SYNPerc
1	0.1.0.0/22	0.0.0.0/2	high	80	6	9.9M	116.4k	11.3%	3.54	53.7%
2	0.1.0.0/24	0.0.0.0/2	*	low	*	8.4M	103.9k	10.1%	3.33	46.4%
3	0.1.0.0/23	0.0.0.0/3	high	low	6	9.8M	113.1k	11.0%	2.84	42.8%
4	0.1.0.0/21	0.0.0.0/3	high	80	6	11.2M	117.1k	11.4%	2.67	44.1%
5	0.1.0.0/23	0.0.0.0/4	high	low	*	8.8M	103.8k	10.1%	2.58	39.7%
6	0.1.0.0/23	0.0.0.0/4	*	low	6	8.7M	104.2k	10.1%	2.54	39.6%
7	0.1.0.0/26	0.0.0.0/2	high	*	6	50.5M	102.9k	10.0%	2.33	9.4%
8	0.1.0.0/20	0.0.0.0/4	high	80	6	10.5M	106.8k	10.4%	2.28	41.8%
9	0.1.0.0/26	0.0.0.0/3	high	*	*	50.5M	104.5k	10.2%	2.18	7.3%
10	0.1.0.0/22	0.0.0.0/6	high	low	6	9.0M	103.0k	10.0%	2.12	35.7%

Table 7: Multidimensional clustering report of Code-Red worm trace.

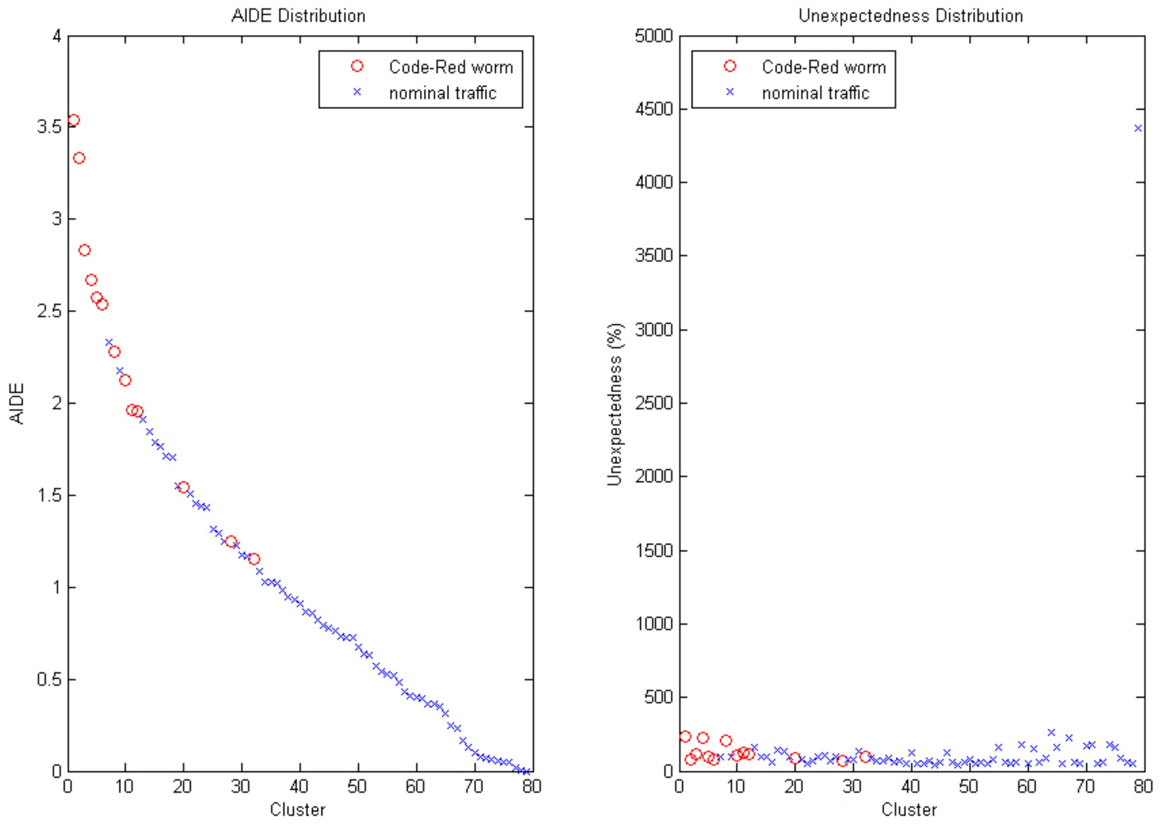

Figure 7: AIDE and unexpectedness distribution of Code-Red worm trace.

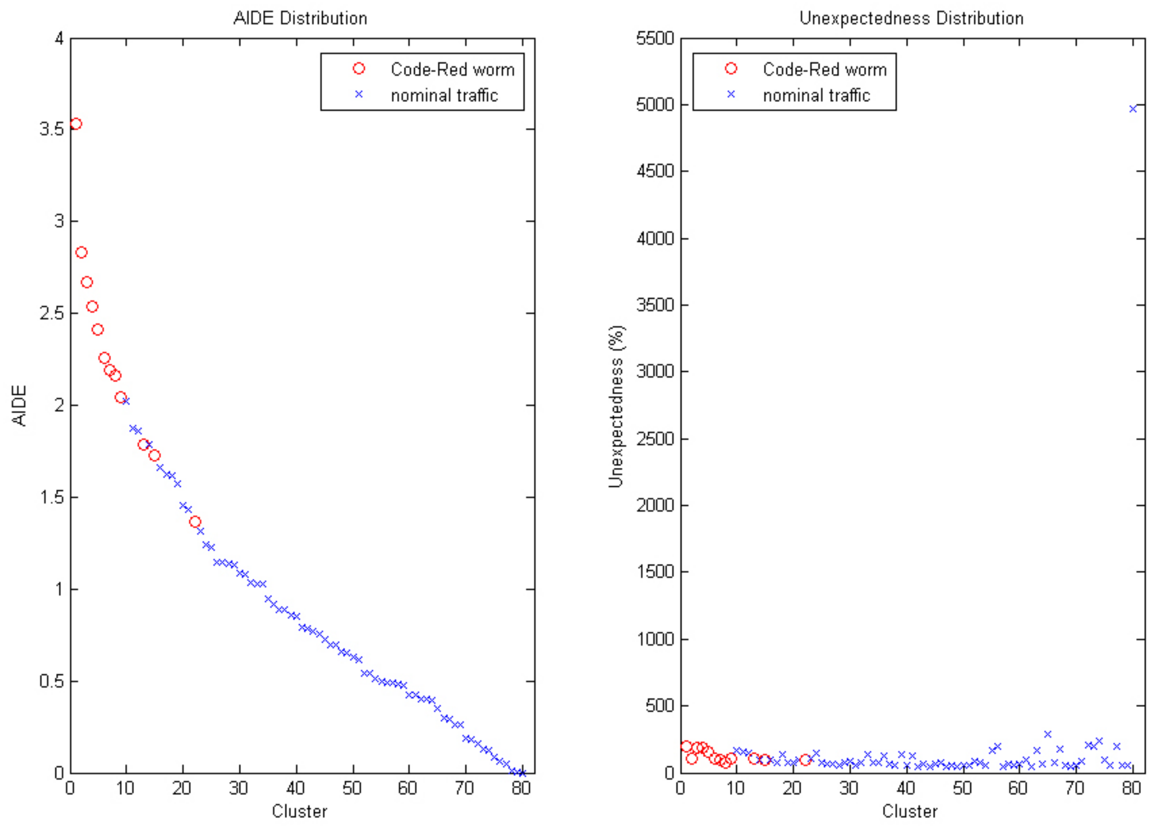

Figure 8: AIDE and unexpectedness distribution of merged Code-Red worm traces.

based on their ports and protocols. Moreover, judging by the five tuples, the second cluster highly overlaps the first cluster; it should also mostly contain worm traffic. Fig. 6 shows the distribution of both AIDE and 'unexpectedness'. Again, only AIDE discriminates attack and nominal clusters.

3). Code-Red version 2 Trace

Code Red (version 2) is a TCP worm that probes random IP addresses on port 80. It spreads much more slowly than a UDP worm such as Slammer. For Code-Red, when a host sends "TCP SYN packets on port 80 to nonexistent hosts", it is considered to be infected [21]. However, because port 80 is a popular one used in TCP (HTTP), it is difficult to *finely* separate worm traffic from nominal traffic, i.e. for clusters that possess worm traffic, it will not in general be possible to isolate the attack packets from the nominal ones. Since this worm is *defined* by the transmission of TCP SYNs, the fraction of SYN packets within a cluster is a very good (if not optimal) feature for discriminating clusters that possess worm from those that do not. However, SYN percentage is only good for identifying TCP worms. We would like to assess whether AIDE (shown previously effective for DDoS and a UDP worm) is also informative about TCP worms.

We define clusters that have (destination port = 80/low, protocol = 6/*) and greater than 20% SYN packets as Code-Red worm clusters. We tested a 1.5-minute trace, obtained at fastigium. The multidimensional report and AIDE/unexpectedness are shown in Table 7 and Fig. 7. Notice the highest AIDE value is not nearly as high as in Table 5 and Table 6 because the top clusters are no longer *pure* attack – they contain much nominal traffic, which decreases AIDE. We also believe that the fact that some (designated) worm clusters have AIDE values lower than those of some nominal clusters again partially stems from the impurity of the worm clusters (they contain non-attack traffic). However, note that most worm clusters still have the highest AIDE values (again, unexpectedness does not distinguish the attack). Since Code-Red does not manifest as quickly as e.g. Slammer, we believe analysis of a longer trace would be more meaningful. However, we only have short Code Red traces. Instead, we merged three traces, obtained from the same router at different times during the worm spread. AIDE and unexpectedness for this merged trace are shown in Fig. 8. By comparing Fig. 7 and Fig. 8, we find AIDE indeed is more effective at discrimination for this longer trace. We also note one can loosely think of this test on the merged trace as a type of collaborative defense experiment (with information pooled over multiple sampling points).

4.2 Discussion and Relation to Prior Work

In [9], the authors recognized the potential application of flow mining to attack identification; however, no experiments were conducted on traces with attacks. The authors also suggested "unexpectedness" to help identify anomalous clusters. It is unclear the independence assumption in this criterion will hold in practice – e.g., one might expect that the source and destination ports are statistically dependent. Our results suggest "unexpectedness" is not as attack-discriminating as AIDE, for traffic with DDoS or worms. It is important to emphasize we are not suggesting to *supplant* existing attack detection techniques. There are already very effective methods for recognizing certain types of attacks, e.g. worm activity by monitoring scans to local dark (unassigned) addresses [4] or by more generally considering failed scan attempts [14][24]. However, our technique can be used to localize the source of the attack, by identifying suspicious clusters and then applying (e.g. existing) techniques solely to the traffic subset fitting a suspicious cluster's definition. Moreover, specifically with respect to slow scanning worms, we propose in the future to monitor the SYN field and correlate traffic coming in both directions while performing digesting, to identify worm clusters as those with great disparity between the number of SYNs and SYN ACKs.

Detailed analysis of mined flows is also a main thrust of [26]. In their approach, one first performs 1-D clustering *separately* for each attribute, similar to the 1-D clustering in [9] and our method, but using an entropy measure, rather than just a traffic volume threshold. The authors then perform several types of analysis on these separate (1-D) flows. In one approach, they measure the entropy for every attribute except the one used to specify the cluster, e.g., for a source IP cluster, the entropy will be measured for the destination IP and both ports. The entropy value is then quantized to a 'low', 'medium', or 'high' range, assigning each source IP cluster to one of 3^4 "behavioral classes". These classes are generally indicative of specific traffic types. The authors also performed "dominant state analysis", seeking to identify strong statistical dependence among attributes.

The most important distinction between [26] and our work is in how IP addresses are treated. [26] treats these attributes as "flat", i.e., they do not consider the hierarchical IP nature. There are two implications. First, their approach cannot define individual clusters that capture a "sector" of IP space. This could be important, e.g., for automatically determining whether a worm is affecting a particular sector of IP space. Second, as aforementioned, the approach in [26] characterizes 1-D clusters by analyzing each attribute *not* defined by the cluster. For our mined clusters, it is possible to follow the authors' approach, measuring the entropy for each "wild-carded" attribute. However, unlike [26], our approach also allows a statistical characterization of attributes that *are* used to define the clusters, i.e. the IP dimensions. Since we treat the IPs hierarchically, for a cluster that specifies, e.g., the first 20 bits of an IP (a 20-bit prefix), we can still give a (conditional) statistical characterization of the remaining 12 bits. In fact, this was actualized in the last subsection, as the entropy used in our AIDE criterion was based on cluster(prefix)-conditional IP distributions. Finally, our method is, in an efficient manner, directly identifying and analyzing *multidimensional* clusters, those specified by value subsets for multiple attributes. Accordingly, as demonstrated in the last section, our approach can give a precise (5 attribute) specification of attacking packets. The approach in [26] discerns attacks separately, from multiple 1-D perspectives. To give a multidimensional description of an attack cluster starting from the 1-D results in [26], one may need to cross-reference and conjoin clusters found for each of the individual dimensions. While this could certainly be done, it is unclear whether this is a computationally modest prospect[16].

[16] Recall that our hierarchical mining algorithm's express purpose, in addition to capturing traffic at different scales, is to efficiently extract the significant multidimensional clusters.

5. Conclusions

We proposed several techniques for improving efficiency of multidimensional flow mining. We also proposed a top-down (unidimensional) method for IP dimensions which removes the need for a large static memory or, alternatively, the need for node creation and sorting (and possibly sampling) operations, while retaining computational efficiency. We compared algorithm execution times using the NZIX trace data. We then demonstrated that our multidimensional flow mining captures, within individual clusters, network attack traffic, including fast scanning worms and DDoS attacks. We suggested and evaluated several criteria for determining whether a cluster is suspicious. Finally, we sketched a methodology for anomaly detection that builds on our flow mining. In future, we will pursue full implementation and evaluation of this methodology.

References

[1] "Mining association rules between sets of items in large databases", R. Agrawal, T. Imielinski, and A. Swami, SIGMOD 1993.

[2] "The space complexity of approximating the frequency moments", N. Alon, Y. Matias, and M. Szegedy, ACM Symp. on the Theory of Computing, pp. 20-29, 1996.

[3] "Detecting early worm propagation through packet matching", X. Chen and J. Heidemann, USC ISI Technical Report ISI-TR-2004-585, 2004.

[4] "Toward Understanding Distributed Blackhole Placement", E. Cooke, M. Bailey, M. Mao, D. Watson, F. Jahanian and D. McPherson, WORM'04, Washington, D.C., Oct. 2004.

[5] "What's new: finding significant differences in network data streams", G. Cormode and S. Muthukrishnan, INFOCOM 2004.

[6] "Diamond in the rough: finding hierarchical heavy hitters in mult-dimensional data", G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava, SIGMOD 2004.

[7] R. O. Duda and P. E. Hart, "Pattern classification and scene analysis", *Wiley*, New York, 1973.

[8] "Charging from sampled network usage", N. Duffield, C. Lund, and M. Thorup, SIGCOMM Internet Measurement Workshop, 2001.

[9] "Automatically inferring patterns of resource consumption in network traffic", C. Estan, S. Savage, and G. Varghese, SIGCOMM2003.

[10] Personal communication with C. Estan.

[11] "Statistical approaches to DDoS attack detection and response", L. Feinstein, D.

Schnackenberg, R. Balupari, and D. Kindred, DARPA Information Survivability Conf. and Expo., 2003.

[12] *Vector quantization and signal compression*, A. Gersho and R. M. Gray, Kluwer, 1992.

[13] "Discovery of multiple-level association rules from large databases", J. Han and Y. Fu, VLDB, 1995.

[14] "Fast portscan detection using sequential hypothesis testing", J. Jung, V. Paxson, A. W. Berger, and H. Balakrishnan, IEEE Symposium on Security and Privacy, May 2004.

[15] "Data streaming algorithms for efficient and accurate estimation of flow size distribution", A. Kumar, M. Sung, J. Xu, and J. Wang, SIGMETRICS/Performance, 2004.

[16] "On the correlation of Internet flow characteristics", K.-C. Lan and J. Heidemann, ISI Technical report ISI-TR-574.

[17] 2000 DARPA Intrusion Detection Scenario Specific Data Sets: Scenario (DDoS) 2.0.2, MIT Lincoln Laboratory, *http://www.ll.mit.edu/IST/ideval/data/2000/LLS_DDOS_2.0.2.html*.

[18] "Mining frequent item sets by opportunistic projection", J. Liu, K. Wang, and J. Han, SIGKDD2002.

[19] "Approximate frequency counts over data streams", G. Manku and R. Motwani, Intl. Conf. on Very Large Databases, 2002, pp. 346-357.

[20] "The Spread of the Sapphire/Slammer Worm", D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, N. Weaver, *http://www.caida.org/outreach/papers/2003/sapphire/sapphire.html*, 2003.

[21] "The Spread of the Code-Red Worm (CRv2)", D. Moore and C. Shannon, *http://www.caida.org/analysis/security/code-red/coderedv2_analysis.xml*, 2001.

[22] Code Red Trace Data, NLANR PMA, *http://pma.nlanr.net/Special/cred.html.*

[23] Slammer/Sapphire Trace Data, NLANR PMA, *http://pma.nlanr.net/Special/slam.html.*

[24] "Very fast containment of scanning worms", N. Weaver, S. Staniford, and V. Paxson, Proceedings of 13th USENIX Security Symposium, August 2004.

[25] *Waikato Applied Network Dynamics Research Group*. Auckland University data traces. *http://wand.cs.waikato.ac.nz/wand/wits/.*

[26] "Profiling Internet Backbone Traffic: Behavior Models and Applications", K. Xu, Z. Zhang and S. Bhattacharyya, Proceedings of ACM SIGCOMM, Philadelphia, PA, August 2005.

[27] "Analysis of the Sapphire Worm – A joint effort of CAIDA, ICSI, Silicon Defense, UC Berkeley EECS and UC San Diego CSE", *http://www.caida.org/analysis/security/sapphire/*, 2003.