Real-time Behavior Profiling for Network Monitoring

Kuai Xu*
Arizona State University
E-mail: kuai.xu@asu.edu
*Corresponding author

Feng Wang
Arizona State University
E-mail: fwang25@asu.edu

Supratik Bhattacharyya
SnapTell Inc
Palo Alto, CA 94306

Zhi-Li Zhang
Department of Computer Science and Engineering
University of Minnesota
E-mail: zhzhang@cs.umn.edu

Abstract: This paper presents the design and implementation of a real-time behavior profiling system for Internet links. The system uses flow-level information, and applies data mining and information-theoretic techniques to automatically discover significant events based on communication patterns. We demonstrate the operational feasibility of the system by implementing it and performing benchmarking of CPU and memory costs using packet traces from backbone links. To improve the robustness of this system against sudden traffic surges, we propose a novel filtering algorithm. The proposed algorithm successfully reduces the CPU and memory cost while maintaining high profiling accuracy. Finally, we devise and evaluate simple yet effective blocking strategies to reduce prevalent exploit traffic, and build a simple event analysis engine to generate ACL rules for filtering unwanted traffic.

Keywords: real-time traffic monitoring; behavior profiling; profiling-aware filtering algorithms.

Biographical notes: Kuai Xu is currently an assistant professor at Arizona State University. He received his Ph.D. degree in computer science from the University of Minnesota in 2006, and his B.S. and M.S. degrees from Peking University, China, in 1998 and 2001. His research interests include network security and cloud computing. He is a member of ACM and IEEE.

Feng Wang received the BS degree from Wuhan University in 1996, MS degree from Beijing University in 1999, and PhD degree from the University of Minnesota in 2005, all in computer science. She is currently an assistant professor at ASU. Her research interests include wireless networks, and combinatorial optimization. She is a member of the IEEE.

Supratik Bhattacharyya received the M.S. and Ph.D. degrees in computer science from the University of Massachusetts, Amherst. He is currently with SnapTell Inc, Palo Alto, CA. He was a Distinguished Member of Technical Staff at Sprint Labs in Burlingame CA. His research interests are in mobile communication and services and in mining network traffic data.

Zhi-Li Zhang received the B.S. degree in computer science from Nanjing University, China, in 1986 and his M.S. and Ph.D. degrees in computer science from the University of Massachusetts in 1992 and 1997. He is currently a Professor of Computer Science and Engineering at the University of Minnesota, where he is currently a Professor. His research interests include computer communication and networks. He is a member of IEEE, ACM and INFORMS Telecommunication Section.

1 Introduction

Recent years have seen significant progress in real-time, continuous traffic monitoring and measurement systems in IP backbone networks (Iannaccone, 2005; Iannaccone et al., 2001). However, *real-time* traffic summaries reported by many such systems focus mostly on volume-based heavy hitters or aggregated metrics of interest (Keys et al., 2005), which are not sufficient for finding interesting or anomalous behavior patterns. This paper explores the feasibility of building a real-time traffic *behavior profiling* system that analyzes vast amount of traffic data in IP backbone networks and reports *comprehensive behavior patterns* of significant end hosts and network applications.

Towards this end, we answer a specific question in this paper: is it feasible to build a *robust* real-time traffic behavior profiling system that is capable of continuously extracting and analyzing "interesting" and "significant" traffic patterns on high-speed Internet links, even in the face of sudden surge in traffic. We address this question in the context of a traffic behavior profiling methodology developed for IP backbone networks (Xu et al., 2005a). The behavior profiling methodology employs a combination of data-mining and information-theoretic techniques to build comprehensive behavior profiles of Internet backbone traffic in terms of communication patterns of end hosts and applications. It consists of three key steps: significant cluster extraction, automatic behavior classification, and structural modeling for in-depth interpretive analysis. This three-step profiling methodology extracts hosts or services that generate significant traffic, classifies them into different *behavior classes* that provide a general separation of various normal and abnormal traffic as well as *rare* and anomalous traffic behavior patterns (see Section 2 for more details). The profiling methodology has been extensively validated *off-line* using packet traces collected from a variety of backbone links in an IP backbone network (Xu et al., 2005a,b).

To demonstrate the operational feasibility of performing *online* traffic behavior profiling on high-speed Internet backbone links, we build a prototype system of the aforementioned profiling methodology using general-purpose commodity PCs and integrate it with an existing real-time traffic monitoring system operating in an Internet backbone network (Xu et al., 2007). The real-time traffic monitoring system captures packets on a high-speed link (from OC12 to OC192) and converts them into 5-tuple flows (based on source IP, destination IP, source port, destination port, protocol fields), which are then continuously fed to the real-time traffic profiling system we build. The large volume of traffic flows observed from these links creates great challenges for the profiling system to process them *quickly* on commodity PCs with *limited memory* capacity. We incorporate several optimization features in our implementation such as efficient data structures for storing and processing cluster information to address these challenges.

After designing and implementing this real-time traffic profiling system, we perform extensive benchmarking of CPU and memory costs using packet-level traces from Internet backbone links to identify the potential challenges and resource bottlenecks. We find that CPU and memory costs linearly increase with flow arrival rate. Nevertheless, resources on a commodity PC are sufficient to continuously process flow records and build behavior profiles for high-speed links in operational networks. For example, on a dual 1.5 GHz PC with 2048 MB of memory, building behavior profiles once every 5 minutes for an 2.5 Gbps link loaded at an average of 209 Mbps *typically* takes 45 seconds of CPU time and 96 MB of memory.

However, resource requirements are much higher under anomalous traffic patterns such as sudden traffic surges caused by denial of service attacks, when the flow arrival rate can increase by several orders of magnitude. We study this phenomenon by superposing "synthetic" packet traces containing a mix of known denial of service (DoS) attacks (Hussain et al., 2003) on real backbone packet traces. To enhance the robustness of our profiling system under these stress conditions, we propose and develop sampling-based *flow filtering* algorithms and show that these algorithms are able to curb steep increase in CPU and memory costs while maintaining high profiling accuracy.

Given the prevalent exploit traffic, we further consider blocking strategies the real-time profiling system can deploy to reduce such unwanted traffic. Based on the characteristics of exploit traffic, we devise several heuristic rules that the profiling system can employ to reducing unwanted traffic, and evaluate their cost and performance. By replaying packet traces collected from backbone links to the real-time profiling system, we find that simple blocking strategies could potentially reduce substantial exploit traffic in backbone networks.

The contributions of this paper are as follows:

- We present the design and implementation of a real-time traffic profiling system for link-level Internet traffic, and demonstrate its operational feasibility by benchmarking CPU and memory costs using packet traces from an operational backbone.
- We propose a new filtering algorithm to improve the robustness of the profiling system against traffic surges and anomalous traffic patterns, and show that the proposed algorithm successfully reduces CPU and memory costs while maintaining high profiling accuracy.
- We devise, evaluate and deploy simple yet effective the access control list (ACL) rules in the real-time profiling system to reduce prevalent exploit behavior in backbone networks.

The remainder of this paper is organized as follow. Section 2 describes a behavior profiling methodology that automatically discovers significant behavior patterns from massive traffic data. Section 3 introduces the real-time profiling system and discusses its functional modules as well as the interfaces with continuous monitoring systems and an event analysis engine. Section 4 is devoted to performance benchmarking and stress test of the profiling system using a variety of packet-level traces from OC-48 backbone links, and synthetic traces that mix various attacks into real backbone packet traces. In Section 5,

*A preliminary version of this paper appeared in the Proceedings of IEEE/IFIP International Conference on Dependable Systems and Networks 2007.

we propose and evaluate sampling-based filtering algorithms to enhance the robustness of the profiling system against sudden traffic surges. Section 6 devises and evaluates several simple blocking strategies in the real-time profiling system to reduce unwanted exploit traffic. Finally, Section 7 concludes this paper.

2 Behavior Profiling Methodology

In light of wide spread cyber attacks and frequent emergence of disruptive applications, we developed a general traffic profiling methodology that automatically discovers significant behaviors with plausible interpretations from vast amount of traffic data. The profiling methodology uses 5-tuple flows, i.e., source IP address (`srcIP`), destination IP address (`dstIP`), source port number (`srcPrt`), destination port number (`dstPrt`), and protocol, collected in a time interval (e.g., 5 minutes) from Internet backbone links. We focus on the first four feature dimensions in 5-tuples, and extract clusters along each dimension. Each cluster consists of flows with the same feature value in a given dimension. The value and its dimension are denoted as *cluster key* and *cluster dimension*. This leads to four groups of clusters, i.e., `srcIP`, `dstIP`, `srcPrt` and `dstPrt` clusters. The first two represent a collection of host behavior, while the last two yield a collection of application behaviors that aggregate flows on the corresponding ports.

2.1 Extracting Significant Clusters

Due to massive traffic data and wide diversity of end hosts and applications observed in backbone links, it is impractical to examine all end hosts and applications. Thus, we attempt to extract *significant* clusters of interest, in which the number of flows exceeds a threshold. In extracting such clusters, we introduced an entropy-based algorithm that finds adaptive thresholds along each dimension based on traffic mix and cluster size distributions.

The intuitive idea of this algorithm is i) to extract clusters in each dimension whose cluster keys are distinct in terms of size distributions; ii) to repeat this process until the size distribution of the remaining clusters in the dimension is (nearly) random. To quantify the "randomness' of cluster size distribution, we use an information-theoretic measure, *relative uncertainty* (also known as standardized entropy), which provides a measure of randomness between 0 and 1. If relative uncertainty (RU) on a given variable X is close to 1, it indicates that the observed values of X are closer to being uniformly distributed and thus look less distinguishable from each other.

By applying this algorithm on a variety of backbone links, we see that the number of significant clusters extracted along each dimension is far less than the total number of values. This observation suggests that this step is very useful and necessary in reducing traffic data for analysis while retaining most interesting behaviors.

2.2 Behavior Classification

Given the extracted significant clusters, the second step of the profiling methodology is to classify their behaviors based on *communication patterns*. The flows in each significant cluster, e.g., a `srcIP` cluster, share the same feature value in `srcIP` dimension, thus most behavior information is contained in the other "free" features including `dstIP, srcPrt, dstPrt`, which might take any possible values.

Traditional approaches mostly focused on volume-based information, e.g., unique number of `dstIP`'s or `dstPrt`'s in examining the patterns of such clusters. However, the traffic volume often is unable to uncover comprehensive communication patterns. For example, if two hosts communicate with 100 unique `dstIP`'s, we cannot safely conclude that their communication patterns from `dstIP` feature are the same without further investigation. A simple example is that one host could be a web server talking to 100 clients, while another is an infected host randomly scanning 100 targets. More importantly, the number of flows associated with each `dstIP` is very likely to be different. For the case of the web server, the numbers of flows between clients and the server tend to be diverse. On the other hand, the number of probing flows between the scanner and each target is often uniform, e.g., one in most cases. This insight motivates us to use relative uncertainty again to measure the feature distribution of free dimensions for all significant clusters.

Suppose the size of a cluster is m and a free dimension X may take N_X discrete values. Moreover, let $P(X)$ denote a probability distribution, and $p(x_i) = m_i/m, x_i \in X$, where m_i is the frequency or number of times we observe the feature X taking the value x_i. Then, the RU in the feature X for the cluster is defined as

$$RU(X) := \frac{H(X)}{H_{max}(X)} = \frac{H(X)}{\log \min\{N_X, m\}}, \qquad (1)$$

where $H(X)$ is the (empirical) *entropy* of X defined as

$$H(X) := -\sum_{x_i \in X} p(x_i) \log p(x_i). \qquad (2)$$

We use relative uncertainty to measure feature distributions of three free dimensions. As a result, we obtain a relative uncertainty vector for each cluster, e.g., [RU_{srcPrt}, RU_{dstPrt} and RU_{dstIP}] for `srcIP` clusters. Recall that RU is in the range of [0,1], so we could represent the RU vector of each `srcIP` cluster as a single point in a 3-dimensional space. Fig. 1 represents each `srcIP` cluster extracted in each 5-minute time slot over an 1-hour period from an OC-48 backbone link as a point in a unit cube. We see that the points are "clustered", suggesting that there are few underlying common patterns among them. Such observation holds for other dimensions as well. This leads to a behavior classification scheme which classifies all `srcIP`'s into *behavior classes* based on their similarity/dissimilarity in the RU vector space.

By applying the behavior classification on backbone links and analyzing their temporal properties, we find this scheme is robust and consistent in capturing behavior similarities among

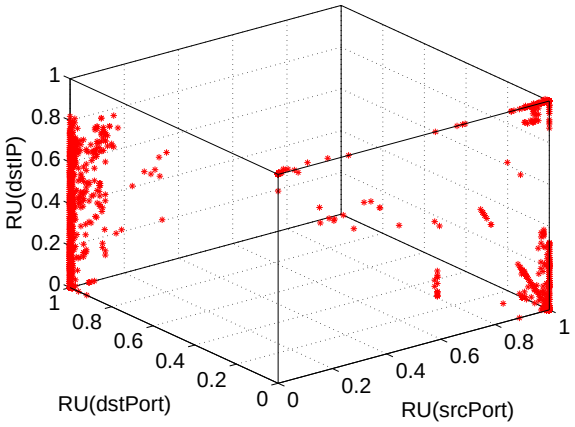


Figure 1: The distribution of relative uncertainty on free dimensions for `srcIP`'s from an OC-48 backbone link during an 1-hour period.

significant clusters. Such similarities are measured on the feature distribution of free dimensions of these clusters, hence provide useful insight in communication patterns of end hosts and applications (Karagiannis et al., 2005; Xu et al., 2005a).

2.3 Structural Modeling

To provide a plausible interpretation for behavior patterns, we adopt *dominant state analysis* technique for modeling and characterizing the interaction of various feature dimensions in a cluster. The idea of dominant state analysis comes from structural modeling or reconstructability analysis in system theory (Krippendorff, 1986; Cavallo and Klir, 1979; Zwick, 2004) as well as more recent graphical models in statistical learning theory (Jordan, 2004).

The intuition behind dominant state analysis is described below. Given a `srcIP` associated with significant traffic flows, all flows can be represented as a 4-tuple (ignoring the protocol field) $\langle u, x_i, y_i, z_i \rangle$, where the `srcIP` has a fixed value u, while the `srcPrt` (X dimension), `dstPrt` (Y dimension) and `dstIP` (Z dimension) may take any legitimate value. Hence each flow in the cluster imposes a "constraint" on the three "free" dimensions X, Y and Z. Treating each dimension as a random variable, the flows in the cluster constrain how the random variables X, Y and Z "interact" or "depend" on each other, via the (induced) *joint* probability distribution $\mathcal{P}(X, Y, Z)$.

The objective of dominant state analysis is to explore the interaction or dependence among the free dimensions by identifying "simpler" subsets of values or constraints (called *structural models* in the literature (Krippendorff, 1986)) to represent the original data in their probability distribution. Given the probability information, we can not only *approximately* reproduce the original flow patterns, but also explain the *dominant* activities of end hosts or applications.

2.4 Properties of Behavior Profiles

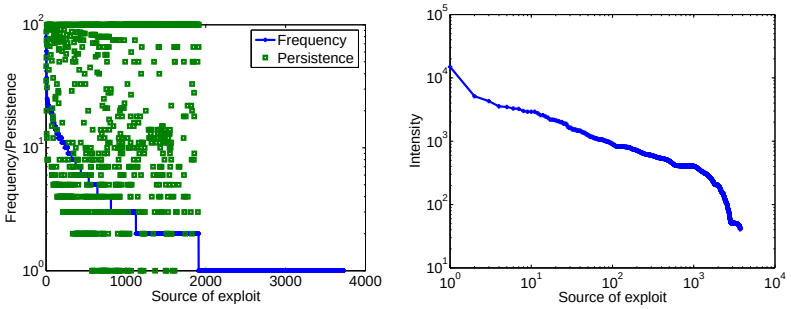
We have applied the profiling methodology on traffic data collected from a variety of links at the core of the Internet through *off-line* analysis. We find that a large fraction of clusters fall into three typical behavior profiles: server/service behavior profile, heavy hitter host behavior, and scan/exploit behavior profile. These behavior profiles are built based on various aspects, including behavior classes, dominant states, and additional attributes such as average packets and bytes per flow. These behavior profiles are recorded in a database for further event analysis, such as temporal properties of behavior classes and individual clusters, or behavior change detection based on RU vectors.

The profiling methodology is able to find various interesting and anomalous events. First, it automatically detects novel or unknown exploit behaviors that match typical exploit profiles, but exhibit unusual dominant states (e.g., `dstPrt`'s). Second, any atypical behavior is worth close examination, since they represent as "outliers" or "anomaly" among behavior profiles. Third, the methodology could point out deviant behaviors of end hosts or applications that deviate from previous patterns.

2.5 Characteristics of Exploit Traffic

Given the prevalent exploit activities, we further introduce several metrics to study the characteristics of exploit traffic. The *frequency*, T_f, measures the number of 5-minute time periods (over the course of 24 hours) in which a source is profiled by our methodology as having an exploit profile. The *persistence*, T_p, measures (in *percentage*) the number of *consecutive* 5-minute periods over the total number of periods that a source sends significant amount of exploit traffic. It is only defined for sources with $T_f \geq 2$. Hence $T_p = 100(\%)$ means that the source continuously sends significant amount of exploit traffic in all the time slots it is observed. Finally, we use the *intensity*, I, to relate both the temporal and spatial aspects of exploit traffic: it measures the (average) number of distinct target IP addresses per minute that a source touches in each 5-minute period. Thus it is an indicator how fast or aggressive a source attempts to spread the exploit.

Figs. 2(a)(b) show the distributions of the frequency vs. persistence and the distribution of intensity for the exploit sources from an OC-48 backbone link during 24 hours. From Fig. 2(a) we observe that frequency follows a Zipf like distribution: only



(a) Frequency (T_f) and persistence (T_p)

(b) Intensity (I)

Figure 2: Characteristics of exploit traffic for the sources with exploit profiles in the backbone link during a 24-hour period.

17.2% sources have a frequency of 5 or more, while 82.8% sources have a frequency of less than 5. In particular, over 70% of them have frequency of 1 or 2. Furthermore, those 17.2% frequent ($T_f \geq 5$) sources account for 64.7%, 61.1% and 65.5% of the total flows, packets, and bytes of exploit traffic. The persistence varies for sources with similar frequency, but nearly 60% of the sources ($T_f \geq 2$) have a persistence of 100 (%): these sources continuously send exploit traffic over time and then disappear.

The exploit intensity illustrated in Fig. 2(b) also follows a Zipf like distribution. The maximum intensity is 21K targets per minute, while the minimum is 40 targets per minute. There are only 12.9% sources with an intensity of over 500 targets per minute, while nearly 81.1% sources have an intensity of less than 500 targets per minute. Those 12.9% aggressive ($I \geq 500$) sources account for 50.5%, 53.3%, and 45.2% of the total flows, packets, and bytes of exploit traffic. The relatively small number of sources, which frequently, persistently or aggressively generate exploit traffic, are candidates for blocking actions.

In the rest of this paper, we will demonstrate the feasibility of designing and implementing a real-time traffic profiling system that uses flow-level information generated from "always-on" packet monitors and reports significant online events based on communication patterns of end hosts and applications even faced with anomalous traffic patterns, e.g., denial of service attacks or worm outbreaks.

3 Real-time Profiling System

In this section, we first describe the design guidelines for the profiling system and then present the overall architecture, functional modules and key implementation details.

3.1 Design Guidelines

Four key considerations guide the design of our profiling system:

- **scalability:** The profiling system is targeted at high-speed (1 Gbps or more) backbone links and hence must scale to the traffic load offered by such links. Specifically, if the system has to continuously build behavior profiles of significant clusters once every time interval T (e.g, T = 5 minutes), then it has to take less than time T to process all the flow records aggregated in every time interval T. And this has to be accomplished on a commodity PC platform.

- **robustness:** The profiling system should be robust to anomalous traffic patterns such as those caused by denial of service attacks, flash crowds, worm outbreaks, etc. These traffic patterns can place a heavy demand on system resources. At the same time, it is vital for the profiling system to be functioning during such events since it will generate data for effective response and forensic analysis. Therefore the system must adapt gracefully to these situations and achieve a suitable balance between profiling accuracy and resource utilization.

- **modularity:** The profiling system should be designed in a modular fashion with each module encapsulating a specific function or step in the profiling methodology. Information exchange between modules should be clearly specified. In addition, the system should be designed to accept input from any packet or flow monitoring system that exports a continuous stream of flow records. However, the flow record export format has to be known to the system.

- **usability:** The profiling system should be easy to configure and customize so that a network operator can focus on specific events of interest and obtain various levels of information about these events. At the same time, it should expose minimal details about the methodology to an average user. Finally it should generate meaningful and easy-to-interpret event reports, instead of streams of statistics.

These design considerations form a guideline of our system design and drive each stage of our system implementation. In the rest of the section, we will discuss the overall architecture of the real-time profiling system, its functional modules and key implementation details that achieve the design goals.

3.2 System Architecture

Fig. 3 depicts the architecture of the profiling system that is integrated with an "always-on" monitoring system, an event analysis engine, and a feedback channel of generating ACL entries of routers for blocking unwanted traffic. The flow-level information used by the profiling system is generated from continuous

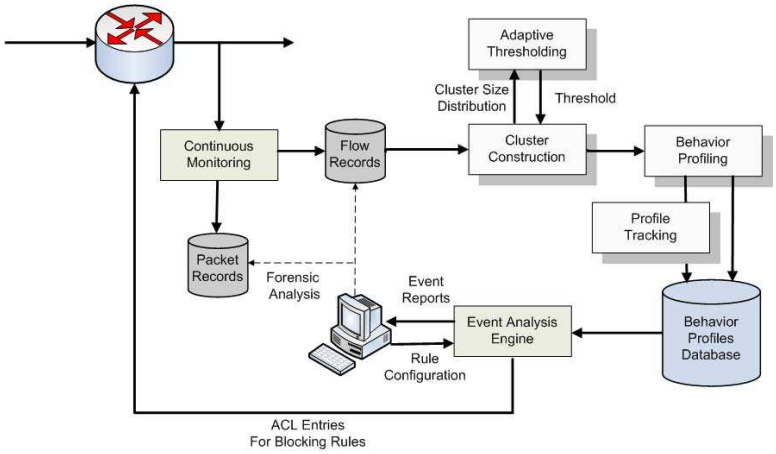


Figure 3: The architecture of real-time traffic profiling system

packet or flow monitoring systems that capture packet headers on a high-speed Internet link via an optical splitter and a packet capturing device, i.e., DAG card. The monitoring system aggregates packets into 5-tuple flows and exports the flow records for a given time interval into disk files. In general, the profiling system could obtain flow records through two ways: shared disk access or file transfer over socket, and the option depends on the locations of the profiling and monitoring systems. The first way works when both systems run on the same machine. If they are located in different machines, the monitoring system transfers the files of flow records in the last time interval to our traffic profiling system. In order to improve the efficiency of the profiling system, we use distinct process threads to carry out multiple task in parallel. For example, one thread continuously reads flow records in the current time interval T_i from the monitoring systems, while another thread profiles flow records that are complete for the previous time interval T_{i-1}.

The real-time traffic profiling system consists of five functional modules (shadowed boxes), namely, "cluster construction", "adaptive thresholding", "behavior profiling" and "profile tracking". Their functions are briefly summarized below:

- **cluster construction:** This module has two initialization tasks. First it starts to load a flow table (`FTable`) in a time interval T into memory from disk files once the profiling system receives a signal indicating `FTable` is ready. The second task is to group flows in `FTable` associated with the same feature values (i.e., cluster keys) into *clusters*.
- **adaptive thresholding:** This module analyzes the distribution of *flow counts* in the four feature dimensions, and computes a threshold for extracting significant clusters along each dimension.
- **behavior profiling:** This module implements a combination of behavior classification and structural modeling that builds behavior profiles in terms of communication patterns of significant end hosts and applications.
- **profile tracking:** This module examines all behavior profiles built from the profiling system from various aspects to find interesting and suspicious network events.
- **event analysis engine:** This module analyzes a *behavior profile database*, which includes current and historical behavior profiles of end hosts and network applications reported by the *behavior profiling* and *profile tracking* modules in the profiling system.

3.3 Key Implementation Details

In this section we focus on some key aspects of our implementation for achieving aforementioned design goals.

3.3.1 Data Structures

High speed backbone links typically carry a large amount of traffic flows. Efficiently storing and searching these flows is critical for the *scalability* of our real-time profiling system. We design two efficient data structures, namely `FTable` and `CTable` for efficient storage and fast lookups during cluster extraction and behavior modeling.

Figure 4 illustrates the data structure of `FTable` and `CTable` with an example. `FTable`, an array data structure, provides an index of 5-tuple flows through a widely-used hashing function, $FH = srcip \hat{} dstip \hat{} srcport \hat{} dstport \hat{} proto \ \% \ (FTableEntries - 1)$, where $FTableEntries$ denotes the maximum entries of `FTable`. For example, in Figure 4, *flow 1* is mapped to the entry 181 in `FTable`, while *flow 2* is mapped to the entry 1. In case of hashing collision, i.e., two or more flows mapping to the same table entry, we use a linked list to manage them. In our experiments, the (average) collision rate of this flow hashing function is below 5% with $FTableEntries = 2^{20}$. While constructing clusters, the naive approach would be to make four copies of 5-tuple flows, and then group each flow into four clusters along each dimension. However, this method dramatically increases the memory cost of the system since the flow table typically has hundreds or millions of flows in each time interval. Instead of duplicating flows, which is expensive, we add four flow pointers (i.e., `next srcIP`, `next dstIP`, `next srcPrt`, and `next dstPrt`) in each flow. Each flow pointer will link the flows sharing the same feature value in the given dimension. For example, the `next srcIP` pointer of *flow 4* links to `flow 3` since they share the same `srcIP`

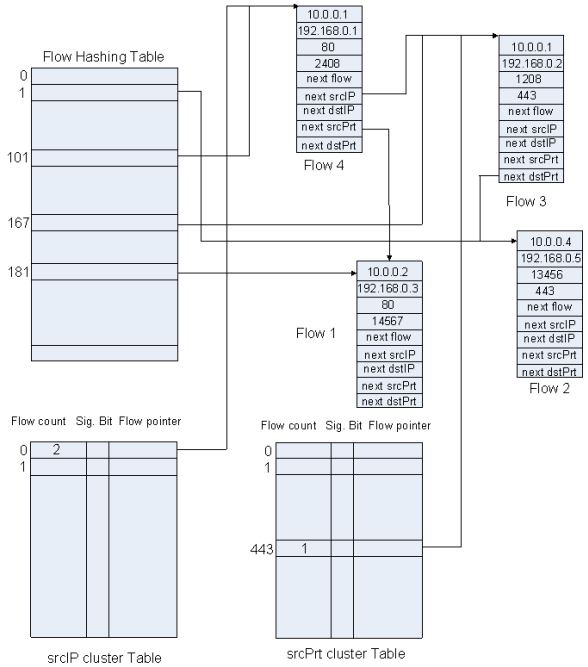


Figure 4: Data structure of flow table and cluster table

10.0.0.1. Similarly, the `next srcPrt` pointer of *flow 4* links to `flow 1` since they share the same `srcPrt` *80*. However, the question is how to quickly find the "old" flows of the same clusters when adding a new flow in the flow table.

To address this problem, we create another data structure, `CTable`, which links the first flow of each cluster in `FTable`. Since there are four types of clusters, we create four instances of `CTable` for managing clusters along four dimensions. Considering `srcPrt` and `dstPrt` dimensions with 65536 possible clusters (ports), we use an array with a size of 65536 to manage the clusters for each of these two dimensions. The index of the array for each port is the same as the port number. For `srcIP` and `dstIP` dimensions, we use a simple hashing function that performs a bitwise exclusive OR (XOR) operation on the first 16 bits and the last 16 bits of `IP` address to map each `srcIP` or `dstIP` into its `CTable` entry. When adding a new flow, e.g., `flow 3` in Fig. 4, in the given `dstPrt`, we first locate the first flow (`flow 2`) of the cluster `dstPrt` 443, and make the `next dstPrt` pointer of `flow 3` to `flow 2`. Finally the first flow of the cluster `dstPrt` 443 is updated to `flow 3`. This process is similar for the cluster `srcPrt` 1208, as well as the the clusters `srcIP` *10.0.0.1* and `dstIP` *192.168.0.2*.

In addition to pointing to the first flow in each cluster, each `CTable` entry also includes flow count for the cluster and significant bit for marking significant clusters. The former maintains flow counts for cluster keys. As discussed in Section 2, the flow count distribution will determine the adaptive threshold for extracting significant clusters.

3.3.2 Space and Time Complexity of Modules

The space and time complexity of modules essentially determines the CPU and memory cost of the profiling system. Thus, we quantify the complexity of each module in our profiling system. For convenience, Table 1 shows the definitions of the notations that will be used in the complexity analysis.

The time complexity of cluster construction is $O(|F| + \sum_{i=0}^{3} |C_i|)$ for `FTable` and `CTable` constructions. Similarly, the space complexity is $O(|F| * s_{fr} + \sum_{i=0}^{3}(|C_i| * r_v))$.

The time complexity of adaptive thresholding is $\sum_{i=0}^{3}(|C_i| * e_i)$. This module does not allocate additional memory, since its operations are mainly on the existing `CTable`. Thus, the space complexity is zero.

The time complexity of behavior profiling is $O(\sum_{i=0}^{3}\sum_{j=0}^{|S_i|}|s_j|)$, while the space complexity is $O(\sum_{i=0}^{3}[|S_i| * (r_b + r_s)])$. The output of this step are the behavior profiles of significant clusters, which are recorded into a database along with the timestamp for further analysis.

Due to a small number of significant clusters extracted, the computation complexity of profile tracking is often less than the others in two or three orders of magnitude, so we will not consider its time and space requirement for simplicity.

3.3.3 Parallelization of Input and Profiling

In order to improve the efficiency of the profiling system, we use *thread* mechanisms for parallelling tasks in multiple modules, such as continuously importing flow records in the current time interval T_i, and profiling flow records that are complete for the previous time interval T_{i-1}. Clearly, the parallelization could reduce the time cost of the profiling system. The disadvantage of doing so is that we have to maintain two set of `FTable` and `CTable` for two time intervals.

Table 1: Notations used in the paper

Notation	Definition
F	set of 5-tuple flows in a time interval
i	dimension id (0/1/2/3 = `srcIP/dstIP/srcPort/dstPort`
C_i	set of clusters in dimension i
S_i	set of significant clusters in dimension i
c_i	a cluster in dimension i
s_i	a significant cluster in dimension i
r_f	size of a flow record
r_v	size of the volume information of a cluster
r_b	size of behavior information of a significant cluster
r_s	size of dominant states of a significant cluster

3.3.4 Event Analysis Engine

To discover interesting or anomalous network events, we build an event analysis engine with three aspects: i) temporal behavior analysis, ii) feature dimension correlation, and iii) event configurations. The objective of temporal behavior analysis is to characterize temporal properties of behavior classes as well as individual clusters from the behavior profile database that records behavior profiles built from the profiling system. Prior work in (Lakhina et al., 2005; Xu et al., 2005a) have demonstrated that temporal properties could help distinguish and classify behavior classes. Feature dimension correlation attempts to find the correlation between clusters from various dimensions to detect emerging exploit and worm activities (Kim and Karp, 2004; Singh et al., 2004) that often trigger new clusters from `srcIP`, `dstIP` and `dstPrt` dimensions.

We develop a simple *event configuration* language that enables network operators or security analysts to extract information on events of interest from behavior profiles for network management or troubleshooting. To express the policy, we use four distinct fields: Dimension, Event Type, Filter, and Description. The options of these fields include:

- Dimension $\in \{srcIP, dstIP, srcPrt, dstPrt, all\}$
- Event Type $\in \{rare, deviant, exploit, unusual\ service\ ports, all\}$
- Filter $\in \{high\ frequncy, high\ intensity, matching\ selected\ ports, others\}$
- Description $\in \{full, summary\}$

For example, if a network operator wants to monitor *rare* behavior of `srcIP` end hosts, she could use the rule srcIP (Dimension) - rare (Event Type) - all (Filter) - full (Description), which expresses the policy of reporting *full* profiles of *all* `srcIP` clusters with *rare* behavior. Similarly, we could construct other filter rules using the combinations of all available options.

In the next section, we will demonstrate the operational feasibility of this system by performing extensive benchmarking of CPU and memory costs using packet-level traces from OC-48 backbone links. To evaluate the robustness of the system, we also test the system against anomalous traffic patterns under denial of service attacks or worm outbreaks.

4 Performance Evaluation

In this section, we first conduct performance benchmarking of CPU and memory cost of the profiling system using a variety of packet traces from OC-48 backbone links. Subsequently, we evaluate the performance bottlenecks of the system under anomalous traffic patterns such as those caused by denial of service attacks and worm outbreaks.

4.1 Benchmarking

We measure CPU usage of the profiling process by using a system call, namely, *getrusage()*, which queries actual system and user CPU time of the process. The system call returns with the resource utilization including *ru_utime* and *ru_stime*, which represent the user and system time used by the process, respectively. The sum of these two times indicates the total CPU time that the profiling process uses. Let T denote the total CPU time, and T_l, T_a, and T_p denote the CPU usage for the modules of cluster construction, adaptive thresholding and behavior profiling, respectively. Then we have

$$T = T_l + T_a + T_p \tag{3}$$

Similarly, we collect memory usage with another system call, *mallinfo()*, which collects information of the dynamic memory allocation. Let M denote the total memory usage, and M_l, M_a, and M_p denote the memory usage in three key modules. Then we have

$$M = M_l + M_a + M_b \tag{4}$$

In oder to track the CPU and memory usages of each module, we use these two system calls before and after the module. The difference of the output becomes the actual CPU and memory consumption of each module. Next, we show the CPU time and memory cost of profiling system on three OC-48 links during a continuous 18 hours with average link utilization of 209 Mbps, 86 Mbps, and 78 Mbps. For convenience, let L_1, L_2, and L_3 denote three links, respectively.

Table 2 shows a summary of CPU time and memory cost of the profiling system on L_1 to L_3 for 18 consecutive hours. It is not surprising to see that the average CPU and memory costs for L_1 are larger than the other two links due to a higher link

Table 2: Total CPU and memory cost of the real-time profiling system on 5-min flow traces

Link	Util.	CPU time (sec)			Memory (MB)		
		min	avg	max	min	avg	max
L_1	207 Mbps	25	46	65	82	96	183
L_2	86 Mbps	7	11	16	46	56	71
L_3	78 Mbps	7	12	82	45	68	842

utilization. Fig. 5 shows the CPU and memory cost of the profiling system on all 5-min intervals for L_1 (the link with the highest utilization). For the majority of time intervals, the profiling system requires less than 60 seconds (1 minute) of CPU time and 150MB of memory using the flow records in 5-min time intervals for L_1.

Fig. 6[a] further illustrates the number of flow records over time that ranges from 600K to 1.6M, while Fig. 6[b] shows the number of all clusters as well as the extracted significant clusters. It is very interesting to observe the similar patterns in the plot of memory cost (Fig. 5[b]) and that of the flow count over time (Fig 6[a]). This observation leads us to analyze the correlation between these two measurements. By examining the breakdown of the memory cost, we find that M_l in the cluster construction module accounts for over 98% of the total memory consumptions. Recall that the space complexity of this module is larger than the others by two or three orders of magnitude, and dominated by the factor in terms of size of flow table $|F|$. The scatter plot of $|F|$ vs. M_l in Fig. 7 reflects the linear relationship between them. Therefore, this strong correlation suggests that the memory cost of the profiling system is mainly determined by the number of flow records collected by the monitoring system in the given time interval.

Fig. 8[a] shows a breakdown in CPU usage of the various modules in the profiling system, and suggests that cluster construction and behavior profiling account for a large fraction of CPU time. Similar to the space complexity, the time complexity in cluster construction is also determined by $|F|$. The linear relationship demonstrated by the scatter plot of $|F|$ vs. T_l in Fig. 8[b] confirms this complexity analysis. Fig. 8[c] shows the scatter plot of the number of significant clusters vs. CPU time in behavior profiling. Overall, we observe an approximately linear relationship between them. This suggests that the CPU cost in behavior profiling is largely determined by the number of significant clusters whose behavior patterns are being analyzed.

To understand how performance is affected by time granularity, we also evaluate the system on L_1 using 1-min, 2-min, 10-min and 15-min flow traces. The results are shown in Table 3. In general, the CPU time and memory cost increase as the length of the time interval. On the other hand, the CPU time of the profiling system is always less than the time interval T. In addition, the average memory cost for 5-min, 10-min and 15-min are 96 MB, 151 MB, and 218 MB, respectively which are within the affordable range on commodity PCs. These results clearly suggest that our real-time profiling system satisfies the *scalability* requirement raised in the previous section.

In summary, the average CPU and memory costs of the real-time profiling system on 5-min flow records collected from an OC-48 link with a 10% link utilization are 60 seconds and 100 MB, respectively. Moreover, the CPU time is largely determined by the number of flow records as well as that of significant clusters, and the memory cost is determined by the number of flow records. During these monitoring periods, these links are not fully utilized, so we can not extensively measure the performance of the real-time profiling system for a highly loaded link. Next, we will test the profiling system during sudden traffic surges such as those caused by denial of service attacks, flash crowds, and worm outbreaks that increase the link utilization as well as the number of flow records.

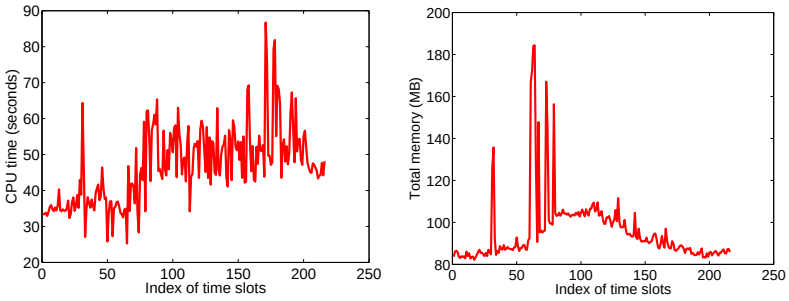
4.2 Stress Test

The performance benchmarking of CPU and memory costs demonstrates the operational feasibility of our traffic profiling system during normal traffic patterns. However, the profiling system should be robust during atypical traffic patterns, such as denial of service attacks, flash crowds, and worm outbreaks (Jung, 2002; Kandula et al., 2005; Moore et al., 2003; Zou et al., 2003). In order to understand the system performance during these incidents, we inject packet traces of three known denial of service attacks and simulated worm outbreaks by superposing them with backbone traffic.

We use the packet traces of three DoS attacks with varying intensity and behavior studied in Hussain et al. (2003). All of these attacks are targeted on a single destination IP address. The first case is a multiple-source DoS attack, in which hundreds of source IP addresses send 4200 ICMP echo request packets per second for about 5 minutes. The second case is a TCP SYN attack lasting 12 minutes from random IP addresses that send 1100 TCP SYN packets per second. In the last attack, a single source sends over 24K *ip-proto 255* packets per second for 15 minutes. In addition to DoS attacks, we simulate the SQL slammer worm on January 25th 2003 (Moore et al., 2003) with an Internet Worm Propagation Simulator used in Zou et al. (2003). In the simulation experiments, we adopt the same set of parameters in Zou et al. (2003) to obtain similar worm simulation results, and collect worm traffic monitored in a 2^{20} IP space.

For each of these four anomalous traffic patterns, we replay packet traces along with backbone traffic, and aggregate synthetic packets traces into 5-tuple flows. For simplicity, we still use 5 minutes as the size of the time interval, and run the profiling system against the flow records collected in an interval. Table 4 shows a summary on flow traces of the first 5-minute interval for these four cases. The flow, packet and byte counts reflect the intensity of attacks or worm propagation, while the link utilization indicates the impact of such anomaly behaviors on Internet links. For all of these cases, the profiling system is able to successfully generate event reports in less than 5 minutes.

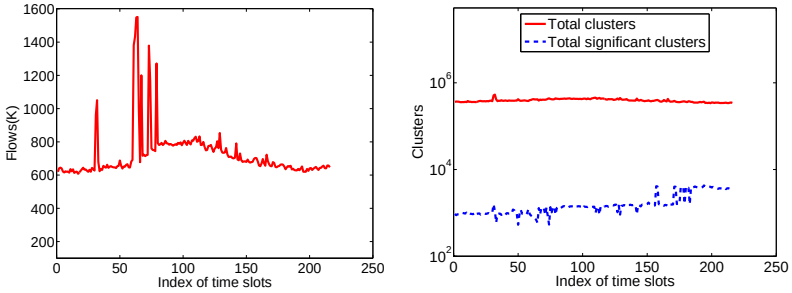
For the synthetic traffic, the link utilization ranged from 314.5 Mbps to 629.2Mbps. We run the profiling system on flow traces after replaying synthetic packets and collect CPU and memory



(a) CPU time (b) Memory cost

Figure 5: CPU and memory cost of the real-time profiling system on flow records in 5-min time interval collected in L_1 for 18 consecutive hours

.



(a) Size of `FTable` (b) Number of clusters

Figure 6: Input of flow traces in 5-min time interval collected in L_1 for 18 consecutive hours

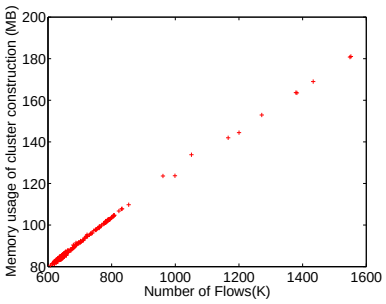
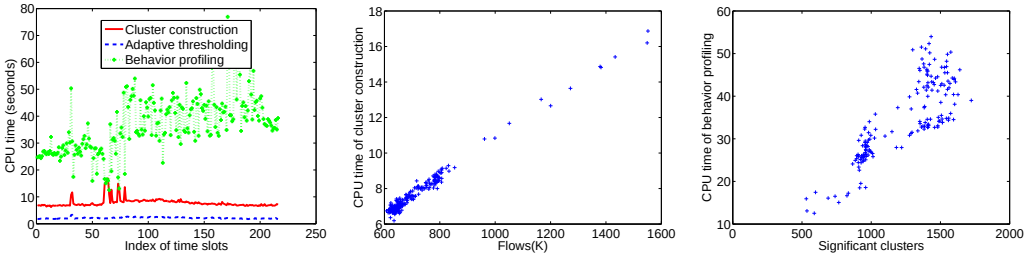


Figure 7: Correlation between memory cost in cluster constructions and size of `FTable`



(a) Breakdown of CPU time (b) Correlation between size of `FTable` and CPU time in cluster construction (c) Correlation between size of `CTable` and CPU time in behavior profiling

Figure 8: Breakdown of CPU time and correlations

Table 3: Total CPU and memory cost with various time granularities

Link	Time scale	CPU time (sec)			Memory (MB)		
		min	avg	max	min	avg	max
L_1	1 min	5	12	31	26	36	76
L_1	2 min	10	22	42	31	46	89
L_1	5 min	25	46	65	82	96	183
L_1	10 min	35	82	129	91	151	208
L_1	15 min	45	88	152	145	218	267

Table 4: Synthetic packet traces with known denial of services attacks and worm simulations

Anomaly	Flows	Packets	Bytes	Link Utilization	CPU time	Memory	Details
DoS-1	2.08 M	18.9 M	11.8 G	314.5 Mbps	45 seconds	245.5 MB	distributed dos attacks from multiple sources
DoS-2	1.80 M	20.7 M	12.5 G	333.5 Mbps	59 seconds	266.1 MB	distributed dos attacks from random sources
DoS-3	16.5 M	39.8 M	16.1 G	430.1 Mbps	210 seconds	1.75GB	dos attacks from single source
Worm	18.9 M	43.0 M	23.6 G	629.2 Mbps	231 seconds	2.01GB	slammer worm simulations

cost of each time interval, which is also shown in Table 4. The system works well for low intense DoS attacks in the first two cases. However, due to intense attacks in the last DoS case (DoS-3) and worm propagations, the CPU time of the system increases to 210 and 231 seconds, but still under the 5 minute interval. However, the memory cost jumps to 1.75GB and 2.01GB indicating a performance bottleneck. This clearly suggests that we need to provide practical solutions to improve the robustness of the system under stress. In the next section, we will discuss various approaches, including traditional sampling techniques and new profiling-aware filtering techniques towards this problem, and evaluate the tradeoff between performance benefits and profiling accuracy.

5 Sampling and Filtering

In this section, we first adopt traditional sampling techniques to address performance bottleneck during sudden traffic surges as caused by severe DoS attacks or worm outbreaks. After evaluating its strength and limitation, we propose a simple yet effective profiling-aware filtering algorithm that not only reduces memory cost, but also retains profiling accuracy.

5.1 Random Sampling

Random sampling is a widely-used simple sampling technique in which each object, flow in our case, is randomly chosen based on the same probability (also known as sampling ratio μ). Clearly, the number of selected flows is entirely decided by the sampling ratio μ. During the stress test in the last section, the profiling system requires about 2GB memory when the number of flow records reach 16.5M and 18.9M during DoS attacks and worm outbreaks. Such high memory requirement is not affordable in real-time since the machine installed with the profiling system could have other tasks as well, e.g., packet and flow monitoring. As a result, we attempt to set 1GB as the upper bound of the memory cost. Recall that in the performance benchmarking, we find that memory cost is determined by the number of flow records. Based on their linear relationship shown in Fig. 7 we estimate that flow records with a size of 10M will require approximately 1GB memory. Thus, 10M is the desirable limit for the size of the flow records.

Using the limit of flow records, l, we could configure the sampling ratio during sudden traffic increase as $\mu = \frac{l}{|F|}$. As a result, we set the sampling ratios in the last DoS attacks and worm outbreaks as 60% and 55%, respectively, and randomly choose flows in loading flow tables in the *cluster construction* module. Table 5 shows the reduction of CPU time and memory consumptions with the sampled flow tables for both cases.

On the other hand, random sampling has substantial impact on behavior accuracy. First, the set of significant clusters from four feature dimensions are smaller than that without sampling, which is caused by the changes of the underlying cluster size distribution after flow sampling. Table 6 shows the number of significant clusters extracted along each dimension without and with sampling for the DoS case. In total, among 309 significant clusters without sampling, 180 (58%) of the *most significant* clusters are extracted with random sampling. Secondly, the behavior of a number of extracted clusters are altered, since flow sampling changes the feature distribution of free dimensions as well as the behavior classes for these clusters. As shown in the last column of Table 6, 161 out of 180 significant clusters with random sampling are classified with the same behavior as those without sampling. In other words, the behavior of 19 (10.5%) extracted significant clusters are changed as a result of random sampling. Fig. 9 shows the feature distributions of free dimensions for 140 `dstIP` clusters with and without random sampling. The deviations from the diagonal line indicate the changes of feature distribution and the behavior due to flow sampling. We also perform random sampling on the synthetic flow traces in the case of worm outbreak, and the results of sampling impact on cluster extractions and behavior accuracy are very similar.

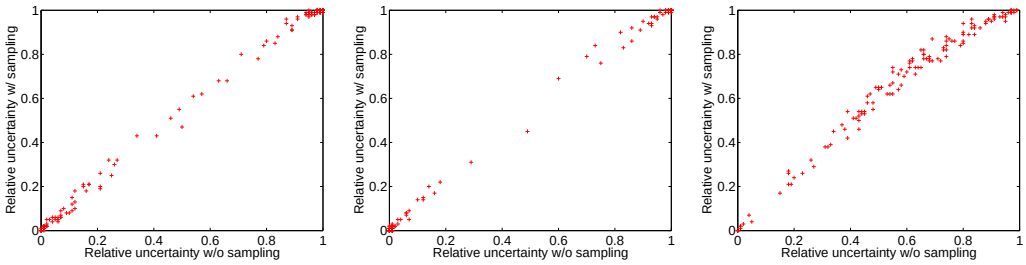
In summary, random sampling could reduce the CPU time and memory cost during sudden traffic surges caused by DoS attacks or worm outbreaks. However, random sampling reduces the number of interesting events, and changes behavior classes of a number of significant clusters. Such impact could have become worse if finer sampling granularities are selected. Thus, it becomes very necessary to develop a profiling-aware algorithm that not only reduces the size of flow tables, but also retains the (approximately) same set significant clusters and their behavior.

Table 5: Reduction of CPU time and memory cost using the random sampling technique

Case	μ	Size of `FTable`	CPU time	memory
DoS attack	66%	10M	89 seconds	867 MB
Worm	55%	10M	97 seconds	912 MB

Table 6: Reduction of significant clusters and behavior accuracy

Dim.	Significant clusters without sampling	Significant clusters with sampling	Clusters with same behavior classes
srcPrt	23	4	3
dstPrt	6	5	4
srcIP	47	31	29
dstIP	233	140	125
Total	309	180	161



(a) `srcPort` (b) `dstPort` (c) `srcIP`

Figure 9: Feature distribution of free dimensions for 140 `dstIP` clusters with and without random sampling

5.2 Profiling-aware Filtering

A key lesson from random sampling is that the clusters associated with denial of service attacks are usually very large in flow count, and hence consume a large amount of memory and CPU time (Argyraki and Cheriton, 2005). In addition, profiling such behavior does not require a large number of flows, since the feature distributions very likely remain the same even with a small percentage of traffic flows. Based on this insight, we develop a profiling-aware filtering solution that limits the size of very large clusters, and adaptively samples on the rest of clusters when the system is faced with sudden explosive growth in the number of flows.

Algorithm 1 describes the details of the profiling-aware sampling algorithm. First, we choose two *watermarks* (L and H) for the profiling system. L represents the moving average of flow tables over time, and H represents the maximum size of flow tables that system will accept. In our experiments, we set $H = 10M$, which is estimated to require 1GB memory cost. In addition, we set the maximum and minimum sampling ratios, i.g., μ_{max} and μ_{min}. The actual sampling μ will be adaptively decided based on the status of flow table size. Specifically, the sampling ratio becomes thinner as the size of flow table increases. For simplicity, let `ftable` denote the size of flow table. If `ftable` is below L, the profiling system accepts every flow. In contrary, if `ftable` is equal to H, the system will stop reading flows and exit with a warning signal.

If `ftable` is equal to L or certain intermediate marks, i.e., $L+i*D$, where D is the incremental factor and $i = 1, 2..., (H-L)/D-1$, the system computes the relative uncertainty of each dimension and evaluates whether there is one or a few dominant feature values along each dimension. In our experiments, we set $D = 1M$ as the incremental factor. The existence of such values suggests that certain types of flows dominate current flow tables, and indicates anomalous traffic patterns. Thus, the system searches these values and marks them as significant clusters for flow filtering. Subsequently, any flow, which contains a feature value marked with *significant*, will be filtered, since such flow will not likely contribute much for the behavior of the associated clusters. Instead, we should give preference to flows that belong to other small clusters. On the other hand, the system could not accept all of these flows with preference after $ftable$ exceeds L watermark. As a result, each of these flows is added with the adaptive sampling ratio

$$\mu = \mu_{max} - i * \frac{\mu_{max} - \mu_{min}}{(H-L)/D - 1}. \quad (5)$$

5.2.1 Evaluations

We run the profiling system on the flow tables in the cases of DoS attack and worm outbreaks (cf. Table 6) with the profile-aware filtering algorithm. Similar to random sampling, this sampling solution could also reduce CPU time and memory cost due to a small size of flow table. However, the profiling-aware sampling has two advantages compared with random sampling. Firstly, the set of clusters extracted using this algorithm is very close to the set without sampling. For example, in the case of DoS attack, the system obtains 41 `srcIP` clusters, 210 `dstIP` clusters, 21 `srcPrt` clusters and 6 `dstPrt` cluster, respectively. Compared with 58% of significant clusters extracted in

Algorithm 1 A Profiling-aware filtering algorithm

Parameters: L, H, D, μ_{max}, μ_{min}, β^*
Initilization: $I = (H - L)/D$
  $\delta_\mu = (\mu_{max} - \mu_{min})/I$
  $\mu = \mu_{max}$
  $i = 0$
  $ftable = 0$
srcPrt = 0; dstPrt = 1
srcIP = 2; dstIP = 3
while next $flow$ **do**
  if ($ftable < L$) **then**
    Insert $flow$ into FTable (Flow Table)
    $ftable++$
    continue;
  else
    if ($ftable > H$) **then**
      Stop reading flows
      exit
    end if
  end if
  if ($ftable == L + i * D$) **then**
    ru_0 = Relative_Uncertainty(FTable, srcPrt)
    ru_1 = Relative_Uncertainty(FTable, dstPrt)
    ru_2 = Relative_Uncertainty(FTable, srcIP)
    ru_3 = Relative_Uncertainty(FTable, dstIP)
    for ($dim = 0$; $dim \leq 3$; $dim++$) **do**
      $ru = ru_{dim}$
      while ($ru \leq \beta^*$) **do**
        remove feature value with highest probability
        mark feature value as $significant$
        ru = Relative_Uncertainty(FTable, dim)
      end while
    end for
    $\mu = \mu_{max} - i * \delta_\mu$
    $i++$
  end if
  if (($ftable \geq L$) && ($ftable \leq H$)) **then**
    if (any feature value in $flow$ is marked $significant$) **then**
      Filter $flow$
    else
      Insert $flow$ into FTable with probability μ
      if ($flow$ is selected **then**
        $ftable++$
      end if
    end if
  end if
end while

random sampling, our profiling-aware algorithm could extract over 90% of 309 original clusters that are selected without any sampling.

Secondly, the behavior accuracy of significant clusters are also improved. Specifically, among 41 `srcIP`'s, 210 `dstIP`'s, 21 `srcPrt`'s, and 6 `dstPrt`'s significant clusters, only 3 `dstIP`'s and 1 `srcPrt` exhibit changes in behavior classes. This finding highly suggests that the profiling-aware profiling algorithm successfully retains the feature distributions of those clusters and behaviors.

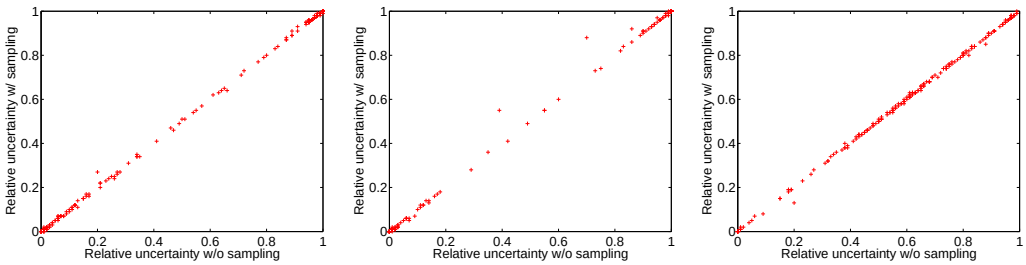
Fig. 10 shows the feature distribution of free dimensions of 210 `dstIP` clusters, extracted both without sampling and with profiling-aware filtering algorithm. In general, the feature distributions of all free dimensions for almost all clusters after filtering are approximately the same as those without sampling. The outliers deviant from the diagonal lines correspond to feature distributions of three clusters whose behavior has changed. Upon close examinations, we find that flows in these clusters contain a mixture of Web and ICMP traffic. The latter are the dominant flows in DoS attacks, so they are filtered after the size of flow table reaches L in the profiling-aware filtering algorithm. The filtered ICMP flows in these clusters explain the changes of the feature distributions as well as the behavior classes.

In the *worm* case, the profiling-aware filtering algorithm also successfully reduces CPU and memory cost of the profiling system, while maintaining high profiling accuracy in terms of the number of extracted significant clusters and the feature distributions of these clusters. Thus, the profiling-aware filtering algorithm can achieve a significant reduction of CPU time and memory cost during anomalous traffic patterns while obtaining accurate behavior profiles of end hosts and network applications.

6 Reducing Unwanted Exploit Traffic

The prevalent exploit behavior indicates the severity of worm or exploit spread and the magnitude of infected hosts (cf. (Yegneswaran et al., 2003; Pang et al., 2004; SNORT, 2009; Paxson, 1999; Staniford et al., 2002)). Given the exploit traffic identified through our real-time profiling system, we devise several heuristic rules of blocking strategies in the profiling system, and then evaluate their efficacy in reducing unwanted traffic (Xu et al., 2005b). In order to determine which sources to block traffic from, we first identify all sources that exhibit exploit behavior. We then devise simple rules to select some or all of these sources as candidates for blocking. Instead of blocking all traffic from the selected sources, we consider blocking traffic on only the ports that a source seek to exploit. This is because exploit hosts may indeed be sending a mixture of legitimate and exploit traffic. For example, if an infected host behind a NAT box is sending exploit traffic, then we may observe a mixture of legitimate and exploit traffic coming from the single IP address corresponding to the NAT box.

For our evaluation, we start with the following benchmark



(a) `srcPort` (b) `dstPort` (c) `srcIP`

Figure 10: Feature distribution of free dimensions for 210 `dstIP` clusters with and without profiling-aware sampling

rule. If a source is profiled as an exploit source during any five minute interval, then all traffic from this source on vulnerable ports is blocked from then on. In general, the benchmark rule could block about 80% of unwanted traffic from these exploit sources. In other words, this rule may still not block all traffic from the source due to two reasons. First, the source might already have been sending traffic, perhaps legitimate, prior to the time-slot in which it exhibited the exploit profile. Second, as explained above, only ports on which we see exploit traffic are considered to be blocked.

While this benchmark rule is very aggressive in selecting sources for blocking, the candidate set of source/port pairs to be added to ACL list of routers may grow to be very large across all links in a network. ACL is a scarce resource of network routers, thus we devise and evaluated other blocking rules that embody more restrictive criteria that an exploit source must satisfy in order to be selected for blocking. We introduce three metrics, *cost*, *effectiveness*, and *wastage* to evaluate the efficacy of these rules. The cost refers to the overhead incurred in a router to store and lookup the ACLs of blocked sources/ports. For simplicity, we use the total number of sources/ports as an index of the overhead for a blocking rule. The effectiveness measures the reduction of unwanted traffic in terms of flow, packet and byte counts compared with the benchmark rule. The resource wastage refers to the number of entries in ACLs that are never used after creations. Based on the evaluations, we find that the rule of blocking aggressive sources (An ACL entry is created if and only if the source has an average intensity of at least 300 flows per minute.) to be very effective.

We apply this blocking strategy and the benchmark rule in the real-time profiling system and replay a 24 hour packet trace collected from an OC-48 Internet backbone link. The benchmark rule achieves the optimal performance, but has the largest cost, i.e., 3756 blocking entries, 34.8% of which are never used after creation. The selected blocking strategy reduces 84.3% unwanted flows, 80.4% unwanted packets, and 72.7% unwanted bytes, respectively. The strategy has a cost of 1789 ACL entries and over 83% of entries are applied at least once to block similar exploit traffic from the same attacker. We believe that that this simple rule is cost-effective when used to block the aggressive or frequent sources that send a majority of self-propagating exploit traffic, in particular, in the early stage of a malware outbreak, to hinder their spread. The similar observations also hold for other Internet backbone links.

In summary, our experiment results demonstrate that blocking the most offending sources in the real-time profiling system is reasonably cost-effective, and can potentially stop self-propagating malware in their early stage of outburst.

7 Conclusions

This paper explores the feasibility of designing, implementing and utilizing a real-time behavior profiling system for high-speed Internet links. We first discuss the design requirements and challenges of such a system and present an overall architecture that integrates the profiling system with always-on monitoring systems and an event analysis engine. Subsequently, we demonstrate the operational feasibility of building this system through extensive performance benchmarking of CPU and memory costs using a variety of packet traces collected from OC-48 backbone links. To improve the robustness of this system during anomalous traffic patterns such as denial of service attacks or worm outbreaks, we propose a simple yet effective filtering algorithm to reduce resource consumptions while retaining high profiling accuracy. Through simple yet effective ACL rules, we demonstrate the application of the real-time profiling system to reduce a significant amount of unwanted exploit traffic.

Acknowledgement: The work is supported in part by the National Science Foundation grants CNS-0626808 and CNS-0626812, a Sprint ATL gift grant, a University of Minnesota DTC DTI grant, and an Arizona State University New College SRCA grant.

REFERENCES

G. Iannaccone (2005), “CoMo: An Open Infrastructure for Network Monitoring - Research Agenda,” Intel Research Technical Report, February 2005.

G. Iannaccone, C. Diot, I. Graham, and N. McKeown (2001), "Monitoring Very High Speed Links," in *Proceedings of ACM SIGCOMM Internet Measurement Workshop*, November 2001.

K. Keys, D. Moore, and C. Estan (2005), "A Robust System for Accurate Real-Time Summaries of Internet Traffic," in *Proceedings of ACM SIGMETRICS*, June 2005.

K. Xu, Z.-L. Zhang, and S. Bhattacharyya (2005), "Profiling Internet Backbone Traffic: Behavior Models and Applications," in *Proceedings of ACM SIGCOMM*, August 2005.

K. Xu, Z.-L. Zhang, and S. Bhattacharyya (2005), "Reducing Unwanted Traffic in a Backbone Network," in *Proceedings of Steps to Reducing Unwanted Traffic on the Internet Workshop (SRUTI)*, July 2005.

K. Xu, F. Wang, S. Bhattacharyya and Z.-L. Zhang (2007), "A Real-time Network Traffic Profiling System," in *Proceedings of International Conference on Dependable Systems and Networks*, June 2007.

A. Hussain, J. Heidemann, and C. Papadopoulos (2003), "A Framework for Classifying Denial of Service Attacks," in *Proceedings of ACM SIGCOMM*, August 2003.

T. Karagiannis, K. Papagiannaki, and M. Faloutsos (2005), "BLINC: Multilevel Traffic Classification in the Dark," in *Proceedings of ACM SIGCOMM*, August 2005.

K. Krippendorff (1986), *Information Theory: Structural Models for Qualitative Data*. Sage Publications, 1986.

R. Cavallo and G. Klir (1979), "Reconstructability Analysis of Multi-dimensional Relations: A Theoretical Basis for Computer-aided Determination of Acceptable Systems Models," *International Journal of General Systems*, vol. 5, pp. 143–171, 1979.

M. Zwick (2004), "An Overview of Reconstructability Analysis," *International Journal of Systems & Cybernetics*, 2004.

M. Jordan (2004), "Graphical models," *Statistical Science, Special Issue on Bayesian Statistics*, vol. 19, pp. 140–155, 2004.

A. Lakhina, M. Crovella, and C. Diot (2005), "Mining Anomalies Using Traffic Feature Distributions," in *Proceedings of ACM SIGCOMM*, August 2005.

H. Kim and B. Karp (2004), "Autograph: Toward Automated, Distributed Worm Signature Detection," in *Proceedings of Usenix Security Symposium*, August 2004.

S. Singh, C. Estan, G. Varghese, and S. Savage (2004), "Automated Worm Fingerprinting," in *Proceedings of Symposium on Operating Systems Design and Implementation*, December 2004.

J. Jung, B. Krishnamurthy, and M. Rabinovich (2002), "Flash Crowds and Denial of Service Attacks: Characterization and Implications for CDNs and Web Sites," in *Proceedings of International World Wide Web Conference*, 2002.

S. Kandula, D. Katabi, M. Jacob, and A. Berger (2005), "Botz-4-Sale: Surviving Organized DDoS Attacks That Mimic Flash Crowds," in *Proceedings of Symposium on NSDI*, May 2005.

D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver (2003), "Inside the Slammer Worm," *IEEE Security and Privacy*, July 2003.

C. Zou, L. Gao, W. Gong, and D. Towsley (2003), "Monitoring and Early Warning for Internet Worms," in *Proceedings of ACM CCS*, October 2003.

K. Argyraki and D. Cheriton (2005), "Active Internet Traffic Filtering: Real-Time Response to Denial-of-Service Attacks," in *Proceedings of USENIX Annual Technical Conference*, April 2005.

V. Yegneswaran, P. Barford, and J. Ullrich (2003), "Internet Intrusions: Global Characteristics and Prevalence," in *Proceedings of ACM SIGMETRICS*, June 2003.

R. Pang, V. Yegneswaran, P. Barford, V. Paxson, and L. Peterson (2004), "Characteristics of Internet Background Radiation," in *Proceedings of ACM SIGCOMM Internet Measurement Conference*, Octomber 2004.

"SNORT," `http://www.snort.org/`.

V. Paxson (1999), "Bro: A System for Detecting Network Intruders in Real-Time," *Computer Networks*, vol. 31, pp. 2435–2463, December 1999.

S. Staniford, J. Hoagland, and J. McAlerney (2002), "Practical Automated Detection of Stealthy Portscans," *Journal of Computer Security*, vol. 10, pp. 105–136, 2002.