A Real-time Network Traffic Profiling System

Kuai Xu, Feng Wang, Supratik Bhattacharyya, and Zhi-Li Zhang*

Abstract

This paper presents the design and implementation of a real-time behavior profiling system for high-speed Internet links. The profiling system uses flow-level information from continuous packet or flow monitoring systems, and uses data mining and information-theoretic techniques to automatically discover significant events based on the communication patterns of end-hosts. We demonstrate the operational feasibility of the system by implementing it and performing extensive benchmarking of CPU and memory costs using a variety of packet traces from OC-48 links in an Internet backbone network. To improve the robustness of this system against sudden traffic surges such as those caused by denial of service attacks or worm outbreaks, we propose a simple yet effective filtering algorithm. The proposed algorithm successfully reduces the CPU and memory cost while maintaining high profiling accuracy.

1 Introduction

Recent years have seen significant progress in real-time, continuous traffic monitoring and measurement systems in IP backbone networks [2]. However, *real-time* traffic summaries reported by many such systems focus mostly on volume-based heavy hitters (e.g., top N ports or IP addresses that send or receive most traffic) or aggregated metrics of interest (total packets, bytes, flows, etc) [6], which are not sufficient for finding interesting or anomalous behavior patterns. In this paper, we explore the feasibility of building a real-time traffic *behavior profiling* system that analyzes vast amount of traffic data in an IP backbone network 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 (OC48 or higher speed) Internet links, even in the face of sudden surge in traffic (e.g., when the network is under a denial-of-service attack)? We address this question in the context of a traffic behavior profiling methodology we have developed for IP backbone networks [9]. 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 *common* "normal" (e.g., web server and service traffic) and "abnormal" (e.g., scanning, worm or other exploit traffic) 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 [9].

To demonstrate the operational feasibility of performing *on-line* 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. 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

*Kuai Xu is with Yahoo! Inc, Feng Wang and Zhi-Li Zhang are with the University of Minnesota, Supratik Bhattacharyya is with SnapTell Inc.

and resource bottlenecks. We find that CPU and memory costs increase linearly with number of flows seen in a given time interval. 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 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 [1] 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.

The contributions of this paper are two-fold:

- We present the design and implementation of a real-time behavior 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.

2 Behavior Profiling Methodology

In light of wide spread cyber attacks and frequent emergence of disruptive applications, we have developed a general traffic profiling methodology that automatically discovers significant behaviors with plausible interpretations from vast amount of traffic data. This methodology employs a combination of data mining and information-theoretic techniques to classify and build behavior models and structural models of communication patterns for end hosts and network applications.

The profiling methodology uses (uni-directional) 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. Since our goal is to profile traffic based on communication patterns of end hosts and applications, 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 collections of host behavior, while the last two yield a collection of port 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 have introduced an entropy-based algorithm [9] that finds adaptive thresholds along each dimension based on traffic mix and cluster size distributions.

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. For example, in a 5-min interval on an OC-48 link, the algorithm extracts 117 significant `srcIP` clusters, 273 `dstIP` clusters, 8 `srcPrt` clusters and 12 `dstPrt` clusters from over a total of 250,000 clusters with the resulting thresholds being 0.0626%, 0.03125%, 0.25% and 1%, respectively. 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 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 features including `dstIP, srcPrt, dstPrt`, which might take any possible values.

Traditional approaches mostly focus 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 [9] to measure the feature distribution of free dimensions for all significant clusters.

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.

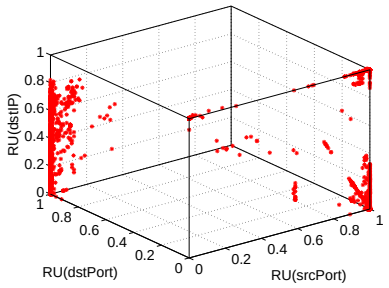


Figure 1. The distribution of relative uncertainty on free dimensions for `srcIP`'s from an OC-48 backbone link during an 1-hour period.

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 significant clusters. Such similarities are measured on the feature distribution of free dimensions of these clusters, hence provide useful insight into communication patterns of end hosts and applications [5, 9].

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 ([11]) as well as more recent graphical models in statistical learning theory [3].

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 [7]) to represent the original data in their probability distribution. Consider a simple example, a `srcIP` cluster consists of 98% scans (with a fixed `srcPrt` 220) to over 1200 random destinations on `dstPrt` 6129. Then the values in the `srcPrt`, `dstPrt` and `dstIP` dimensions these flows take are of the form $\langle 220, 6129, * \rangle$, where $*$ (wildcard) indicates random values. Clearly this cluster is dominated by the flows of the form $\langle 220, 6129, * \rangle$. We refer to such forms as *states* of a cluster. Hence given the information about the states, 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.

To summarize, the profiling methodology has demonstrated the applicability of the profiling methodology to critical problem of detecting anomalies or the spread of unknown security exploits, profiling unwanted traffic, and tracking the growth of new applications. However, the practical value of the profiling framework largely depends on the operational feasibility of this system in a *real-time* manner.

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

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, 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 varying 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.

3.2 System Architecture

Fig. 2 depicts the architecture of the profiling system that is integrated with an "always-on" monitoring system and an

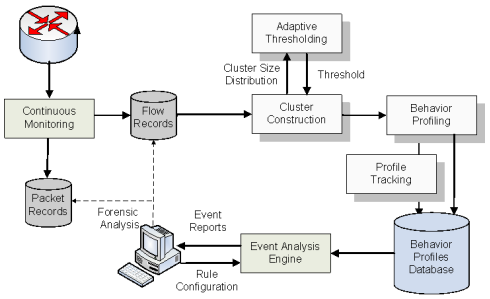


Figure 2. The architecture of real-time traffic profiling system

event analysis engine. The flow-level information used by the profiling system are generated from continuous 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 obtains flow records through three ways: i) shared disk access, ii) file transfer over socket, and iii) flow transfer over a streaming socket. The option in practice will depend on the locations of the profiling and monitoring systems. The first way works when both systems run on the same machine, while the last two can be applied if they are located in different machines.

In order to improve the efficiency of the profiling system, we use distinct process threads to carry out multiple task in parallel. Specifically, 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 event analysis engine 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.

The real-time traffic profiling system consists of four functional modules (shadowed boxes), namely, "cluster construction", "adaptive thresholding", "behavior profiling" and "profile tracking". Each of these modules implements one step in the traffic profiling methodology described in Section 2.

3.3 Key Implementation Details

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.

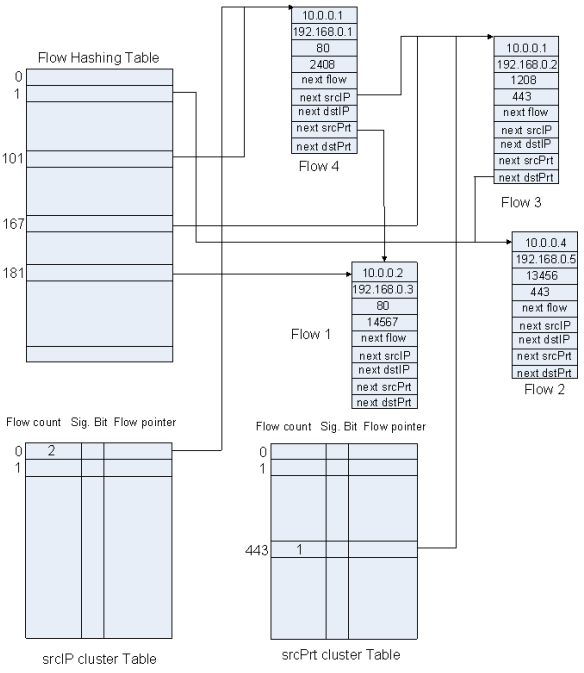


Figure 3. Data structure of flow table and cluster table

We design two efficient data structures, namely `FTable` and `CTable` for efficient storage and fast lookups during cluster extraction and behavior modeling.

Figure 3 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 commonly-used hash 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 3, *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 hash 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` *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 hash 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. 3, 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.

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 sig. cluster
r_s	size of dominant states of a significant cluster

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

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 for simplicity we will not consider its time and space requirement.

3.3.3 Parallelization of Input and Profiling

In order to improve the efficiency of the profiling system, we use *thread* mechanisms for parallelizing 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 consecutive time intervals.

4 Performance Evaluation

4.1 Benchmarking

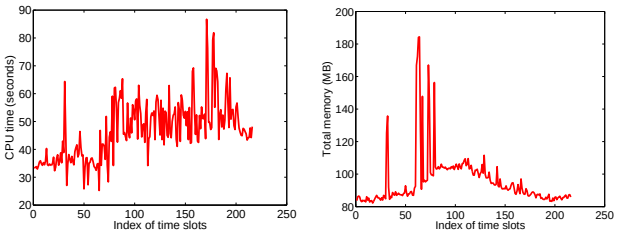
We measure CPU usage of the profiling process by using a system call, namely, *getrusage()*, which queries actual system and user CPU time of the process. The system call returns with the resource utilization including *ru_utime* and *ru_stime*, which represent the user and system time used by the process, 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

$$T = T_l + T_a + T_p \quad (1)$$

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

$$M = M_l + M_a + M_b \quad (2)$$

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-hour period with an average link utilization of 209 Mbps, 86 Mbps, and 78 Mbps. For convenience, let L_1, L_2, and L_3 denote these three links, respectively.



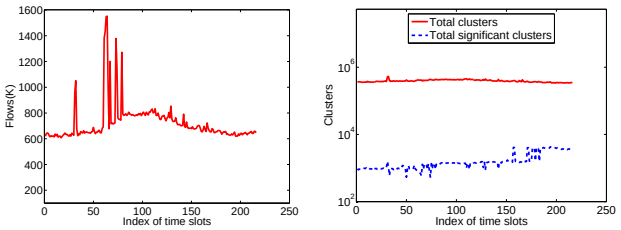
(a) CPU time (b) Memory cost

Figure 4. 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

.

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 utilization. Fig. 4 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. 5[a] further illustrates the number of flow records over time that ranges from 600K to 1.6M, while Fig. 5[b]



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

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

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. 4[b]) and that of the flow count over time (Fig 5[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 size of flow table $|F|$. A deep examination on $|F|$ vs. M_l confirms 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 a given time interval.

The breakdown in CPU usage 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 confirms this complexity analysis. In addition, we observe an approximately linear relationship between the number of significant clusters and CPU time in behavior profiling. This suggests that the CPU cost in behavior profiling is largely determined by the number of significant clusters whose behavior patterns are being analyzed.

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 increases the link utilization as well as the number of flow records.

4.2 Stress Test

The performance benchmarking of CPU and memory costs demonstrate the operational feasibility of our traffic profiling system during normal traffic patterns. However, the profiling system should also be robust during atypical traffic patterns, such as denial of service attacks, flash crowds, and worm outbreaks [4, 8, 10]. 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 [1]. 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 with 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 [8] with an Internet Worm Propagation Simulator used in [10]. In the simulation experiments, we adopt the same set of parameters in [10] 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 3 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.

During the emulation process, 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 cost of each time interval, which is also shown in Table 3. 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,

Table 3. 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

Table 4. 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

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

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 18.9 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 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.

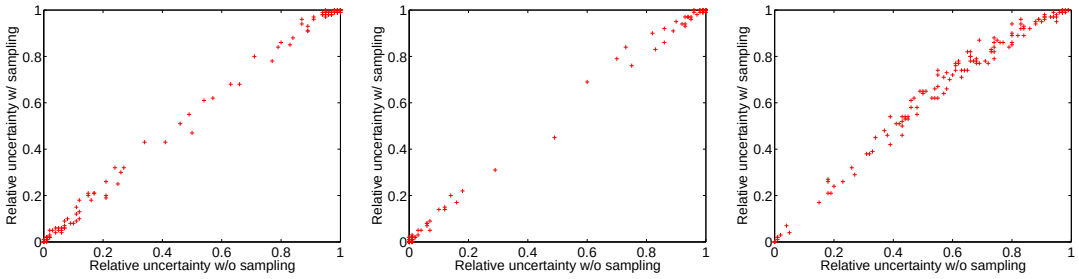
Using the limit of flow records, l, we could configure the sampling ratio during sudden traffic increase as $\mu = \frac{l}{|F|}$. As a result, we set the sampling ratios in the last DoS attacks and worm outbreaks as 60% and 55%, respectively, and randomly choose flows in loading flow tables in the *cluster construction* module. Table 4 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 5 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 still 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 5, 161 out 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. 6 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.

Table 5. Reduction of significant clusters and behavior accuracy

Dim.	Sig. clusters without sampling	Sig. 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

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 also alters the behavior classes of some significant clusters. Such impact could have become worse if "lower" sampling rates are selected. Thus, it becomes necessary to develop a profiling-aware algorithm that not only reduces the size of



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

Figure 6. Feature distribution of free dimensions for 140 `dstIP` clusters with and without random sampling

flow tables, but also retains the (approximately) same set significant clusters and their behavior.

5.2 Profiling-aware Filtering

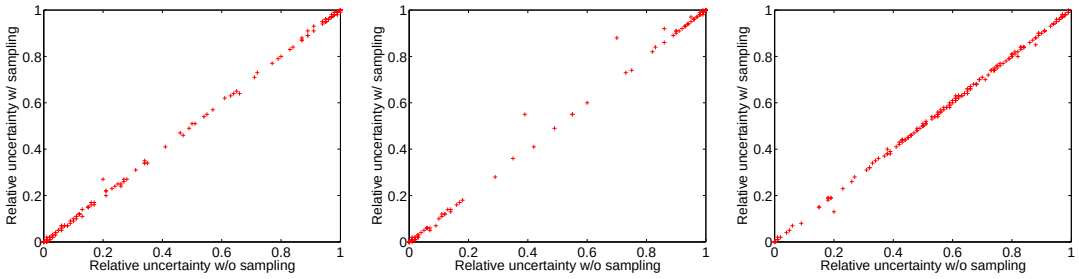
A key lesson from random sampling is that the clusters associated with denial of service attacks are usually very large in flow count, and hence consume a large amount of memory and CPU time. In addition, profiling such behavior does not require a large number of flows, since the feature distributions 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.

The details of the profiling-aware sampling algorithm are as follow. 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 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 affect the behavior of the associated clusters. On the other hand, additional flows for other small clusters have substantial contributions to their behavior. Thus, we should give preference to flows that belong to such 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 $\mu = \mu_{max} - i * \frac{\mu_{max} - \mu_{min}}{(H-L)/D-1}$.

We run the profiling system on the flow tables in the cases of DoS attack and worm outbreaks with the profile-aware filtering algorithm. Like random sampling, *profiling-aware* sampling also reduces CPU time and memory cost by limiting the size of flow table. On the other hand, the profiling-aware sampling has two advantages over the random sampling. First, 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 random sampling, our profiling-aware algorithm could extract over 90% of 309 original clusters that are selected without any sampling. Second, 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` clusters change to "akin" classes from their original behavior classes. These findings suggest that the *profiling-aware* profiling algorithm approximately retains the feature distributions of significant clusters and behaviors.

Fig. 7 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



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

Figure 7. Feature distribution of free dimensions for 210 `dstIP` clusters with and without profiling-aware sampling

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 Conclusions and Future Work

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. We are currently in the process of integrating the event analysis engine into a rule-based anomaly detection system. In addition, we are extending the flow import/export protocol so that the profiling system could work with multiple continuously monitoring systems. Finally we would like to correlate anomalous and interesting events from multiple monitoring points.

Acknowledgement: This work was supported in part by the NSF grants CNS-0435444 and CNS-0626812, a University of Minnesota Digital Technology Center DTI grant and Sprint ATL gift grant.

References

[1] A. Hussain, J. Heidemann, and C. Papadopoulos. A Framework for Classifying Denial of Service Attacks. In *Proceedings of ACM SIGCOMM*, August 2003.

[2] G. Iannaccone, C. Diot, I. Graham, and N. McKeown. Monitoring Very High Speed Links. In *Proceedings of ACM SIGCOMM Internet Measurement Workshop*, November 2001.

[3] M. Jordan. Graphical models. *Statistical Science, Special Issue on Bayesian Statistics*, 19:140–155, 2004.

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

[5] T. Karagiannis, K. Papagiannaki, and M. Faloutsos. BLINC: Multilevel Traffic Classification in the Dark. In *Proceedings of ACM SIGCOMM*, August 2005.

[6] K. Keys, D. Moore, and C. Estan. A Robust System for Accurate Real-Time Summaries of Internet Traffic. In *Proceedings of ACM SIGMETRICS*, June 2005.

[7] K. Krippendorff. *Information Theory: Structural Models for Qualitative Data*. Sage Publications, 1986.

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

[9] K. Xu, Z.-L. Zhang, and S. Bhattacharyya. Profiling Internet Backbone Traffic: Behavior Models and Applications. In *Proceedings of ACM SIGCOMM*, August 2005.

[10] C. Zou, L. Gao, W. Gong, and D. Towsley. Monitoring and Early Warning for Internet Worms. In *Proceedings of ACM CCS*, October 2003.

[11] M. Zwick. An Overview of Reconstructability Analysis. *International Journal of Systems & Cybernetics*, 2004.