Nabs: A System for Detecting Resource Abuses via Characterization of Flow Content Type

Kulesh Shanmugasundaram
kulesh@isis.poly.edu

Mehdi Kharrazi
mehdi@isis.poly.edu

Nasir Memon
memon@poly.edu

Polytechnic University
Brooklyn, NY11201

Abstract

One of the growing problems faced by network administrators is the abuse of computing resources by authorized and unauthorized personnel. The nature of abuse may vary from using unauthorized applications to serving unauthorized content. Proliferation of peer-to-peer networks and wide use of tunnels makes it difficult to detect such abuses and easy to circumvent security policies. This paper presents the design and implementation of a system, called `Nabs`, that characterizes content types of network flows based solely on the payload which can then be used to identify abuses of computing resources. The proposed method does not depend on packet headers or other simple packet characteristics hence is more robust to circumvention.

1 Introduction

Abuse of computing resources is one of the growing problems. Network administrators deal with a variety of abuses such as, network bandwidth by unauthorized application services, and the distribution of unauthorized content to name a few. Abusers can be malicious attackers looking for free resources to host their illegal activities, a malicious insider running a peer-to-peer hub, or simply an ill informed user unintentionally running an application proxy.

The two most common defenses that are used to prevent network abuses are firewalls and Intrusion Detection Systems (IDS). An IDS is not useful in detecting many types of abuses where the essence of the abuse is not captured by a simple set of signatures. Firewalls, on the other hand, are more effective in preventing abuse. Firewalls use port blocking to thwart unauthorized application services. For instance, if a security policy denies the use of web servers inside a network then a firewall simply blocks traffic to port 80.

However, it is now well known that a firewall can be circumvented. For example, most firewalls do not block outbound connection requests. A malicious insider or a host inside the network compromised by an attacker can initiate a connection and transfer unauthorized data or make available an unauthorized service without being detected by a firewall. Another simple way to bypass the firewall would be to simply run the unauthorized service on a port that the firewall allows traffic on. So for example, if the firewall blocks services on port 80 and leaves port 22 open so that users can telecommute, then a web server can be configured to use port 22, thereby circumventing the security policy. A third way to get past the firewall is by tunneling. Tunneling works by encapsulating a network protocol within packets carried by another protocol. So in the above example, with the presence of a suitable proxy on the inside host, web traffic could be tunneled through SSH traffic on port 22. Similarly, there are many other techniques to get past a firewall, given a malicious insider or a captured host inside the target network.

Firewall circumvention techniques give rise to new challenges in abuse detection. Current state-of-the-art in abuse detection is to simply use port blocking or bandwidth throttling. Routers simply monitor the bandwidth usage of hosts and enforce throttling when it exceed a preset limit. This is not always an effective solution as the bandwidth may be used for legitimate purposes. There have been some research work in identifying application types in the presence of weak port binding [4, 13]. However, we believe the knowledge of content carried by network flows gives better granularity on detecting abuse more robustly than the methods presently used. Therefore, the method proposed in this paper does not rely on packet header information for *Flow Content Characterization*. By flow content characterization we mean the ability to classify network packet contents as belonging to one of a set of data types like audio data, encrypted data, video data etc. Note that our intention is not to identify the application being used but to identify the type of content emanating from a host or a network flow in general.

A naive method to characterize flow content is to simply use the media headers of various file types,

like the `file(1)` command on Unix systems. Such as approach has many problems. First of all, media headers (like the JPEG headers or MPEG headers) can be modified easily. Therefore, it is easy to circumvent a method that relies on the header information. On the other hand, not every single packet contains header information. For instance, suppose there is a 200KB JPEG image. When transmitted over network this image will be split into around 200 packets and only one of them contains the header. A header based monitoring system must be able to examine each packet on the network for the string "JFIF" to determine the content type is a JPEG image. Such a method will also result in false positives as the string "JFIF" could appear in a JPEG image or in a text file. In order to minimize such false positives the method would require some context information be maintained to properly identify the text. This is obviously a very expensive process in terms of processing power and memory and such an approach is not viable on large networks where traffic volume is high. Besides, packet drops and asymmetric routing may result in this method losing the packet that has the header information rendering it useless. Also note that some media types do not have headers at all. For example, plain-text and encrypted content usually have no headers to indicate their content type. Hence, the proposed method does not rely on media headers. The method samples packets from a network, groups them into flows and uses the group of packets to characterize the flow content based on its statistical properties.

The rest of the paper is organized as follows: in the following section we present an overview of `Nabs`. Subsequent sections, Section 3, Section 4, and Section 5, discuss various components of the system in detail. In Section 6 we discuss deployment and experiences gained while running this system in a live network. Related work is presented in Section 7 and we conclude with a summary of current accomplishments and future work in Section 8.

2 Overview of `Nabs`

`Nabs` is a tool developed for characterizing the content types of flows. Information about content types of flows can significantly improve various applications such as, resource provisioning, QoS policy developments, traffic accounting, and billing. Furthermore, a system like `Nabs` also has the potential to be an intrusion detection system. In this paper, however, we focus on its application for detecting abuse of resources. Abuse can be defined as an act considered unacceptable by the community sharing resources. In the presence of a use policy, which formally defines acceptable acts, abuse can be defined precisely as any deviation from the use policy. Use policies are currently defined using parameters like bandwidth usage, port numbers, and type of applications. For instance, a use policy may state that *a user's net bandwidth limit is 2GB per month*. This leads to inconveniences to users who use up the bandwidth to download legitimate content. Besides, these parameters can be easily manipulated hence use policies that rely on such parameters are easily subverted. Flow content characterization allows us to incorporate content types into use policies. It is more difficult to manipulate content types because of the difficulty in changing the underlying statistical properties of one content type to another without distorting the original content. Therefore, incorporating content types makes use policies more expressive and robust against subversion. Now, we can restate the above use policy as *a user's net bandwidth on multimedia content (audio, video, images) is limited to 2GB per month*, which is more user friendly.

`Nabs` is currently deployed in our network and monitors all TCP, UDP flows. Figure 1 illustrates an overview of the system's design. It collects network packets and groups them into flows, characterizes the content of each flow, and stores the results for future use. It has three major components to achieve these tasks, which we briefly discuss in the rest of this section.

Flow Collection & Throttling. For the purposes of this paper, a flow is defined as a set of packets that have identical quine-tuple $Protocol, SourceIP, DestinationIP, SourcePort, DestinationPort$. The flow collection component sniffs the network and captures all traffic passing through a monitoring point. Packet capture and filtering is accomplished using `libpcap` and BPF filters [2]. Packets that pass through the filter are then grouped into flows and scheduled in the flow-table to be picked up for flow characterization.

Flow Characterization. Flow characterization component constantly sweeps the flow-table for flows that have accumulated necessary data (16KB of payload, for example). When such a flow is found it is subsequently removed from the flow-table and processed to identify the type of its content. Output from flow characterization, of the form `<time, flow-id, flow-type [,auxiliary-data]>`, is then stored in a database or used to answer queries in real-time as described below. `flow-id` is the concatenation of quine-tuple mentioned above, `flow-type` is the content type of the flow as determined by the classifier, and `auxiliary-data` includes number of packets and bytes in the flow.

Storage. Output from the classifier can be stored directly to a database. However, the storage space required to store this data outweighs the usefulness of

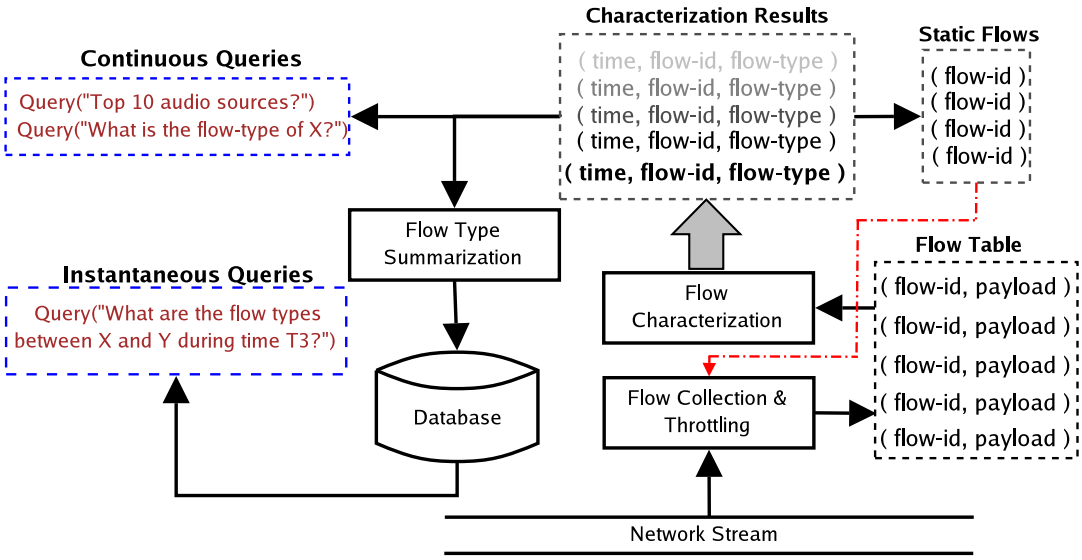

Figure 1: Architecture of The Proposed System

such fine grained data. Storage requirements can be eased by summarizing the results without loosing too much information in the process. Currently we have implemented a simple summarization technique that merges duplicate {`flow-ids, flow-type`} pairs from the output and stores the resulting data set.

User Interaction & Query Processing. A user can extract necessary information from the database via a user interface using a SQL-like query language. Query processing has two major components. One of them is dedicated to *continuous queries* and the other to *instantaneous or one-time queries*. Continuous queries process the characterization results as a stream and updates the results in real-time. Continuous queries are useful for monitoring networks in real-time for such information as "*What are the top-k sources of audio in the network now?*" or "*What are the type of flows emanating from host x now?*" Instantaneous queries, on the other hand, are carried out on data stored in the database and are useful for analysis of events postmortem.

3 Flow Collection & Throttling

Flow characterization requires a certain minimum amount of payload per flow to determine the content type. Until the required payload is accumulated flows are buffered in the flow-table. This arrangement necessitates a garbage collector which prevents flows that do not carry the required minimum from occupying precious memory. Two major factors stand in the way of optimizing memory utilization. First, small insignificant flows take up valuable space in the flow-table preventing interesting flows from being buffered. Second, even among the interesting flows we may not need to look at every single packet to characterize the flow. Therefore, packets must go through a mechanism that throttles packets based on a preset strategy. Please note that the system is a passive monitor and when we say "throttle" we mean throttling flows entering the system and *not* throttling the flows themselves. We now describe the throttling strategy currently used.

Throttling & Lossy Counting. Throttling flows would require us to keep track of flow rates (packets per second or bandwidth) of all flows entering the system. Naive approach of building a table to keep track of flows consumes too much memory. What we need is an efficient and flexible way of measuring flow rates. Over the years various data structures and algorithms have been developed for this purpose. For our implementation we choose one such algorithm, lossy counting [6], for the following reasons:
Deterministic: Among the many probabilistic algorithms lossy counting is one of the few deterministic algorithms that can maintain an ϵ-deficient synopsis of data within the error bounds specified by user.
Streaming Algorithm: It is an one-pass algorithm which means lossy counting computes the necessary information on a single pass over the data. It is, therefore, well suited for network streams.
Efficiency & Footprint: It is computationally efficient requiring only a few additions and comparisons per packet. The data structure is easy to maintain, has small memory footprint, and self pruning.

We now briefly describe lossy counting and refer the readers to [6] for a detailed description and analysis of the algorithm and to [8] for an excellent survey on data stream algorithms in general.

Lossy Counting. Lossy counting is a streaming algorithm that can deterministically compute approximate frequency counts of elements exceeding a user-specified threshold in a space efficient manner. More formally, suppose N denotes the length of current stream and s, ϵ are two user-specified parameters support and error respectively, then lossy counting estimates the frequency of elements in the stream whose true frequency exceeds sN with the guarantee that the estimated frequencies are utmost ϵN less than the true frequencies by using utmost $\frac{1}{\epsilon}\log(\epsilon N)$ space.

The algorithm works as follows: the data stream is conceptually divided into *buckets* of width $w = \lceil \frac{1}{\epsilon} \rceil$ elements each. Buckets are labeled with *bucket ids* starting with 1 and let *current bucket id* be $b_{current}$ whose value is ϵN. A table $\mathcal{D}$ of rows of the form (e, f, Δ) is maintained where e is the element, f is the frequency, and Δ is the value of $(b_{current} - 1)$ when e was inserted into the table. Initially $\mathcal{D}$ is empty. Whenever an element e arrives , the algorithm first looks up table $\mathcal{D}$ to see if the element is listed. If so, then frequency f is incremented by one for the corresponding entry. Otherwise, an entry of the form $(e, 1, b_{current}-1)$ is inserted into $\mathcal{D}$. Table $\mathcal{D}$ is pruned at bucket boundaries, whenever $N \equiv (0 \ mod \ w)$, by removing entries where $f + \Delta \leq b_{current}$. Note that for an entry e in $\mathcal{D}$, f denotes the exact frequency of e ever since it is inserted into $\mathcal{D}$. Now to find elements exceeding threshold s we simply walk through entries in $\mathcal{D}$ and extract entries where $f \geq (s - \epsilon)N$.

We use lossy counting to keep track of the flow rate of each flow and at bucket boundaries we obtain a list of flows which exceeds the user-specified threshold s. All flows that do not satisfy this threshold are discarded and those that satisfy the threshold are put into the flow-table. For example, setting $\epsilon = 0.001$ and $s = 0.01$ would result in flows that exceed 1% of total traffic be placed in the flow-table. Since lossy counting has no false negatives none of the flows above 1% will be missed. However, flows that are between 0.9% and 1% might or might not appear in the stream and are false positives. This is a good trade-off between accuracy and resources as we will never miss flows that we are interested in (above 1% of total traffic) and will eliminate most of the flows that we are not interested in (below 0.9%) but only incur some overhead in processing the false positives (between 0.9% and 1%).

The above throttling technique and packet filtering together gives us better control over precisely which flows should be monitored by the system. For instance, we have the flexibility to specify something like "*consider only TCP or UDP flows to or from ports above* 1024 *that occupy more than* 1% *of total traffic.*" This is a considerable advantage when monitoring traffic on large networks.

4 Flow Characterization

This component is responsible for determining content types of flows buffered in the flow-table. In order to distinguish between a variety of flow content types, we looked at the payload of each packet as a vector of bytes. Thus, our goal was to come up with a model that would help us distinguish these vectors based on their respective statistical signatures. The statistical measures we used to build the model can be grouped into three broad categories:

Time Domain. We choose a number of simple statistical measures from the time domain. Although some of these measures are simple and rudimentary, they help greatly in distinguishing content types. These measures were, mean, variance, auto-correlation, and entropy. For example one would expect that RAW data formats such as, bitmap images, or .WAV audio, to have lower entropy than compressed or encrypted formats. This is evident in Figure 2, which shows the average entropy of data fragments for 1000 files in each of the 8 major content types. A discussion of the data set used can be found in Section 4.1. Similar reasoning justifies the use of variance and auto-correlation as well.

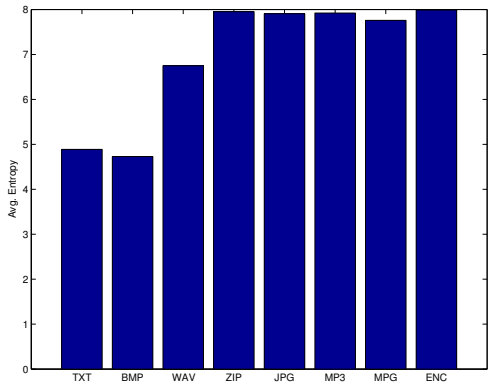

Figure 2: Average entropy of data vectors from 8 different file types.

Frequency Domain. Inspecting the frequency domain representation of a set of byte vectors obtained from different types of files we noticed subtle differences in frequency representations depending on the original data type, we choose to use a number of statistical measures from the frequency spectrum. We first divided the frequency spectrum into 4 bands ranging from, $0-\pi/8$, $\pi/8-\pi/4$, $\pi/4-\pi/2$, and $\pi/2-\pi$. We then calculated the mean, variance, power, and skewness of each band. For example the average mean of the power in the $0-\pi/8$ band of the frequency spectrum can be seen in Figure 3.

Higher Order Statistics. Finally, we looked at bicoherence, which is a third order statistic. The bico-

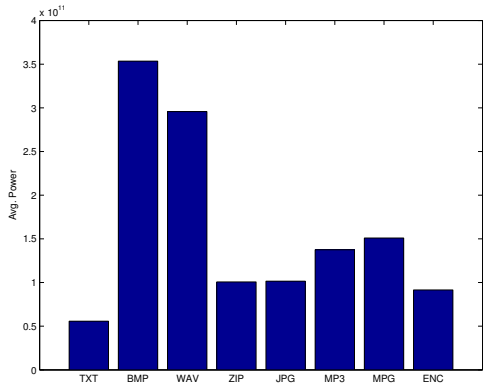

Figure 3: Average mean of the first band, $0 - \pi/8$, of the power spectrum for data vectors from 8 different file types.

herence is able to characterize non-linearities in the underlying data. Our argument is that the amount of non-linearity introduced by the compression or encryption techniques varies. Thus these measures could help us distinguish these content types. We first computed the bicoherence, after which power of the bicoherence magnitude and phase, and the mean of the bicoherence magnitude are calculated. In addition to these statistics we also computed the kurtosis and skewness of each byte vector. For a good review of bicoherence and more generally higher order statistics the reader is referred to [7].

4.1 Offline Experiments

In this section we describe experiments carried out to determine three critical parameters related to the classifier. First, we would like to determine the effectiveness of the features we have in distinguishing the content type. Then, we would like to find the appropriate trade-off between the required minimum data for classification and accuracy of classification. Finally, we would like to determine trade-off between the number of features used for classification and accuracy so that we can increase the through put of the flow characterization component. We begin this section with the explanation of the data set followed by the experimental setup.

Data Set. There are a variety of content types available on the Internet. One could divide these content types into three major categories: raw (or uncompressed), compressed, and encrypted data. Our goal was to evaluate how well we can distinguish between data from each category. We selected a number of different content types from each category. For example, in the raw category we looked at content types of plain-text, BMP, and WAV, and in the compressed category ZIP, JPG, MP3, and MPEG files. Our data set, consisting of the 7 different content types, was obtained from a random crawl of a peer-to-peer network. The only constraint placed on the downloads was that the files be at least 50KB. A total of 1000 files were downloaded for each file type. These files were then encrypted using the AES encryption algorithm to obtain 1000 encrypted files.

Classification. There are a variety of classification algorithms available. We have chosen to use Support Vector Machines [14] in our experiments based on our previous experiments with a number of different data sets, and observing consistently better performance results over other classifiers. In our experiments we opted to use the RBF kernel (Radial Basis Function). The RBF kernel was optimized by doing a grid search over its two parameters: cost and gamma. There are many implementations of SVM available on the public domain and we have chosen the freely available LibSvm [3] for our experiments and for implementation in the `Nabs` itself.

Experimental Setup. The proposed statistics were computed over various sizes of payload. In order to simulate sampling packets off the wire we segmented each file into 1024-byte blocks. The 1024-byte block was chosen in accordance with the average size of TCP packet with payload. 32768 bytes of data, or equivalently 32 packets were collected from random locations in each file. Since we were interested in seeing the effects of the size of available data on the classification results, we then obtained different size payloads from the 32768 bytes of sampled data. Given a payload size, statistics were then computed for the payload. After which we have 1000 feature vectors containing the identifying features proposed for each of the eight categories. A SVM classifier was then trained using 400 feature vectors from each of the 8 content types. The remaining 600 feature vectors were then used to test the resulting classifier.

4.2 Results

The above procedure was repeated for different payload sizes, and as expected the accuracy of the classification improved with the size of payload used for classification. In our experiment accuracy was defined as:

$$Accuracy = \frac{T}{T+F} \tag{1}$$

where T is the number of samples classified correctly, and F is the number of samples classified incorrectly. Figure 4 shows the results of accuracy vs. payload size trade-off. Interestingly, we observed that the accuracy begins to saturate as the features are computed over payloads larger than 16KB.

Since we were building a multi-class classifier, just looking at the overall accuracy would not give a com-

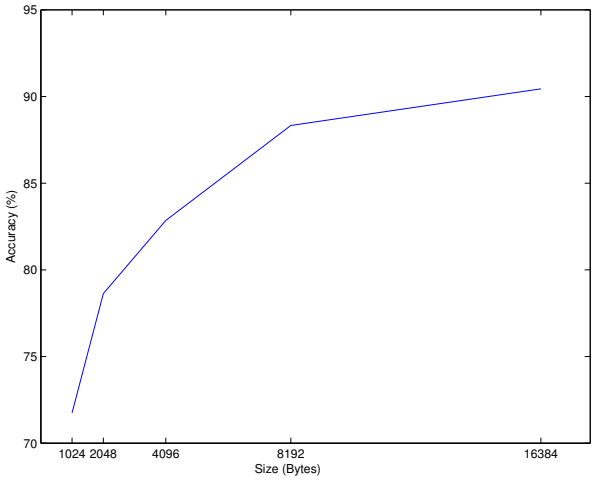

Figure 4: Accuracy of classification for various payload sizes.

plete picture on how well the classifier was able to distinguish between the categories. In order to see how well each category is distinguished with respect to the others, we computed the confusion matrix. Confusion matrix presents information about the actual and predicted results using the classifier. These entries should not be misinterpreted as accuracy figures. In fact, the overall accuracy of the classifier is equal to the average of the diagonal entries in the confusion matrix. In Table 4.2 we show the confusion matrix for payloads of size 16KB. From the table we can observe that 96.33% of plain-text payloads were classified correctly. However, 2.83%, 0.17%, 0.67%, and 0.17% of them were misclassified as BMP, WAV, ZIP, and MPG respectively.

4.3 Feature Selection

Some of the features described in the feature set above may have very little or no information gain in distinguishing between the different categories. Furthermore, when implementing the actual system, speed and complexity become an issue so one would only want to employ the more essential features from the 25 proposed features. Therefore, we used SFFS (Sequential Forward Feature Selection) [11] algorithm to identify and extract the essential features. This algorithm sequentially adds or removes features and finds the best subset of features which give maximal information gain in the classification process. As seen in Figure 5, we could obtain optimal accuracy by using only 6 of 25 features. In fact the accuracy has less than 1% of difference with the case that all 25 features are used. The chosen features are in order of importance entropy, power in the first frequency band, mean, variance, mean and variance in the fourth frequency band.

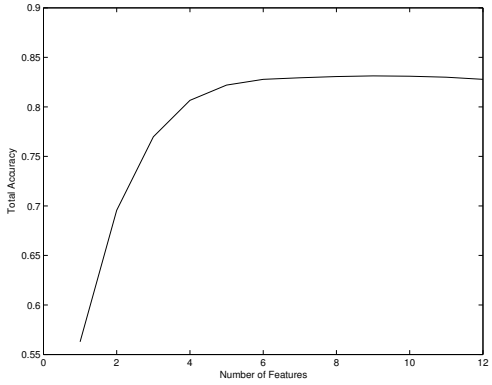

Figure 5: Accuracy of classification for number of features selected by SFFS.

Using the selected features, the classifier was retrained and tested using the data segments of size, 1024, 4096, 8192, and 16384 bytes. Detection results can be seen in Figure 6. As evident from the figure, although feature selection provides only marginal improvement, it greatly reduces required processing power per flow. Thus with only 6 features we are able to obtain similar results as if we were using all 25 features.

5 Flow Scheduling

As we can observe from the previous section, flow characterization is slower than flow collection and on large networks the characterization becomes a bottleneck. Network flows can be categorized into four major groups based on packet rate over time (sustained, temporary) and content type (static, dynamic.)

Sustained Static Flows. These flows have constant packet rate for long periods of time, several minutes or hours. These flows do not change the content type during their life time. Sustained static flows are generally the result of streaming audio/video or downloading a large file (like an ISO image).

Sustained Dynamic Flows. Similar to the above in terms of packet rate and lifetime but the content type of the flow changes with time. Example of such flows include, accessing network file systems and downloading files via a file sharing program.

Temporary Static Flows. Temporary flows are mostly bursts of traffic that lasts only for a few seconds or perhaps minutes utmost. Most network traffic is of this form– web requests and emails to name a few. These flows carry a single type of content.

Temporary Dynamic Flows. Lifetime of the flow is same as above but the content type changes. The change in content type is due to the fact a file may have various embedded contents. Examples of such

	Predicted							
	Txt	Bmp	Wav	Zip	Jpg	Mp3	Mpg	Enc
Txt	**96.33**	2	0.67	0.83	0	0.17	0	0
Bmp	2.83	**91**	3.67	1.17	0.5	0.17	0.5	0.17
Wav	0.17	3.17	**88.33**	1.33	0.83	5.67	0.5	0
Zip	0.67	0	0.17	**73.83**	6	1.5	1.17	16.67
Jpg	0.17	1.33	0.83	4.17	**89.83**	2	1.67	0
Mp3	0	0.67	1.17	0.83	0.83	**95.83**	0.67	0
Mpg	0.83	2.33	0.83	0.67	2	2.67	**90.67**	0
Enc	0	0	0	2.33	0	0	0	**97.67**

Table 1: Confusion matrix for flow content characterization using payload of size 16384 bytes (or roughly equivalent to using payloads from 16 packets).

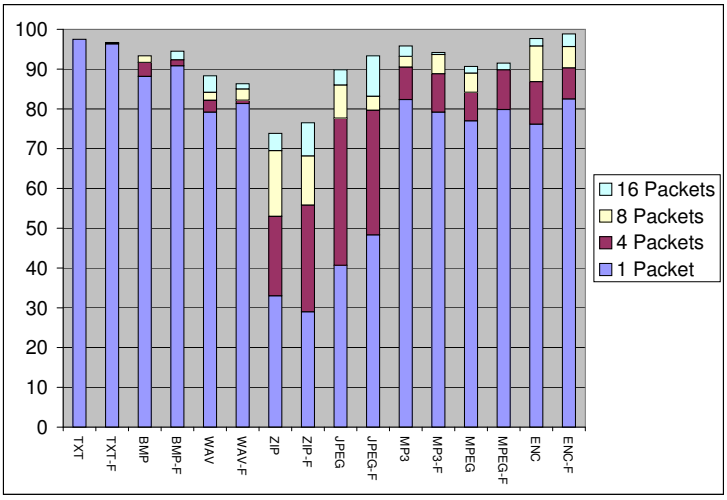

Figure 6: Characterization results using all 25 features and the results using the selected 6 features (marked with a -F at the end of the category name) for each category.

flows include a flow of downloading a Postscript file with embedded image or downloading a Microsoft Word document.

It is easy to see that sustained flows will occupy the classifier most of the time even though repeatedly characterizing them yields no additional information. This is especially true for sustained static flows as the flow content does not change at all. Even for sustained dynamic flows the content type change is gradual enough to skip a few packets in the flow. Proposed scheduling identifies sustained flows (regardless of dynamic or static) and throttles them from entering the characterization component. This prevents dominating flows from using the classifier repeatedly and balances the use of expensive classification across all four types of flows.

Scheduling Sustained Flows. Within the framework of lossy counting we first define sustained flows more formally. Given a threshold s, a flow is considered sustained if it was above the threshold in the past n buckets where n is a user specified parameter. Now we describe the scheduling policy used in `Nabs`.

Given the definition for sustained flows, the algorithm to identify sustained flows works as flows: input to the algorithm is the stream of output from the flow characterization component. For each element in the output stream a table $\mathcal{D}$ is updated as in lossy counting. In addition, if the corresponding entry satisfies $(b_{current} - \Delta) \geq n$ then the entry is removed from the table and the `flow-id` is sent to the flow collection and throttling component. The throttling component prevents packets corresponding to the flow from entering the flow-table for a preset epoch k. This epoch

could be a time interval or a packet count. In our implementation $k = n$ hence throttling will prevent the flow from entering the flow-table for next n buckets. Flow-ids that have been throttled for the past k buckets are then removed from the throttling component which then allows the corresponding flows to use the classifier. This cycle continues until either the flow falls below threshold s or it is finished.

Given that the flow characterization is done multiple times on a single flow, it is possible to characterize the content type mix in a flow. For example, we can state that a particular flow is composed of "20% audio, 45% video, and 35% compressed" content. We call such an estimation *flow composition*. Obviously when the scheduler throttles flows it is bound to affect the accuracy of flow composition. Note, however, this scheduling only affects the flow composition of sustained dynamic flows. Neither temporary flows nor sustained static flows are affected by the above throttling because temporary flows are not throttled at all and content type of sustained static flows does not change hence flow composition does not apply. In the worst case, scheduling policy drops *all* packets of a set of content types hence flow compositions of these types, as observed by the system and end user, are zero. This happens when a content type begins every n bucket and lasts for the next k buckets. In the best case, packets from each content type will be dropped proportional to their contribution to the flow hence the actual and observed flow composition are identical. On average, however, the accuracy depends on the ordering of packets in the flow– which in turn depends on network and application latency and routing delays. In the next section we show that the scheduling is still feasible and does not affect the flow composition accuracy much.

6 Deployment & Experiences

Now we summarize our experiences running the system on a live network with hundreds of active hosts for two weeks. `Nabs` was deployed at a traffic concentration point of our campus network to monitor all TCP and UDP flows. The OC3 link on average carried 10.57MB/s and was utilized 55% of its capacity. The system ran on a 3GHz Pentium IV with 1GB of RAM equipped machine with Linux Kernel 2.4.25. Flow characterization component was set to use 16KB of data for characterizing content types. The garbage collector in the flow-table was set to remove flows that could not accumulate 16KB in 60 seconds. During the two week monitoring period we observed `Nabs` processing about 600 flows per second on average. Flow characterization on 16KB payload, including computing the 6 features on the payload and classifying the content types using SVM, took $945\mu s$. During the monitoring period the system's average main memory usage was 15MB and never exceeded 20MB.

6.1 Use Policy & Abuse Detection

Although the system can identify the content type of any flow to tag an event or set of events as abuse we must first define a use policy to detect abuses precisely. We summarize the results of observing `Nabs` in the network during the two-week period.

Use Policy 1. Encrypted Content Encrypted traffic is allowed only for the purposes of remote shell and secure web transactions. No other form of encrypted traffic is allowed within the network or to outside hosts from within the network.
Abuse: We found plenty of hosts in the network being sources of encrypted traffic however most are legitimate uses. Upon closer examination, we found 9 hosts on two subnets being sources of significant amount of encrypted traffic. Nmapping the hosts reveal them being part of Waste peer-to-peer network which, encrypts connections between the nodes.

Use Policy 2. Multimedia Content Besides the designated web servers, hosts within the network shall not serve multimedia content.
Abuse: To locate the abusers of this policy we queried the system for hosts emanating content types MP3, MPEG, or JPEG. Among the many hosts, we found 16 hosts of interest based on the amount of multimedia traffic they serve. Further investigation revealed them running proxy servers such as http-proxy and ccftp-proxy, and contents emanate from these proxies. We also found some machines running Internet Relay Chat servers. One of the hosts is running QMQP server which, implements Quick Mail Queuing Protocol, and used by Spammers for mail relaying.

Use Policy 3. MP3 Trollers. Web trollers are crawlers that targets web servers and ftp servers extracting specific contents from the servers, such as images and MP3 files. This policy disallows the use of any trollers against servers in the network.
Abuse: Designated web servers did not emanate any MP3 or MPEG traffic therefore we believe the trollers are useless against these servers.

The above results are very promising in that `Nabs` is able to detect these abuses where firewalls and intrusion detection systems deployed in the network failed.

6.2 Flow Composition

Finally, we wanted to determine the accuracy of flow composition observed by the system. For this experiment, we used four content types, namely MP3,

MPEG, JPEG, and plain-text, and constructed ten different flows with random permutations of these types. Size of each flow was 86MB. We then sent the streams across the network and monitored it using `Nabs`. Table 6.2 shows the actual and observed flow composition. Observed composition is averaged over all ten flows.

	Audio	Video	Text	Jpeg
Actual	24.83%	25.70%	24.41%	24.83%
Observed	27.46%	28.08%	19.75%	20.93%
Difference	2.63%	2.38%	4.66%	3.90%

Table 2: Actual and observed flow composition of a 86MB flow with four different types of content.

Even though the flow was only 86MB due to the scheduler between the classifier and flow-table only half of the flow was examined by the classifier and rest is dropped by the scheduler to accommodate other flows on the network. On average, `Nabs` examined only 47.52MB or 55.25% of the total flow. As we can see the results are promising in that they accurately depict the flow composition of a relatively short-lived flow. Note that the observed flow composition does not add up to 100%. This is due to false positives from the classifier which sometimes, for example, miss-classifies JPEG as compressed (gzip or zipped) content. We believe, `Nabs` will perform even better on longer flows.

6.3 Caveats of The System

We believe the following issues must be raised in the context of flow characterization for the sake of completeness and for motivating future research in this area.

Pipelining. Connection pipelining is the process of sending multiple requests/resources through a single connection. Pipelining of connections may result in various types of content interlaced in an application depended manner. This could increase the false positives of the classifier. However, we did not observe applications using pipelining.

DataMasking. In [9] Radhakrishnan et. al. propose a method to change the statistical properties of encrypted data to that of non-encrypted content. We are not clear on the practicality of such method. DataMasking, however, could be a potential threat to flow content characterization if it were feasible.

Compression. Zip/Gzip compressed content is often confused with other compressed content types such as MP3, JPEG, and MPEG. We believe this is due to the fact that compression is not perfect and leaves traces of the statistical properties of the underlying data.

Privacy. We are not addressing the issue explicitly but it is an important one to consider when implementing the system. However, it can be easily incorporated into the system by means of proper authorization mechanisms and monitoring policies.

6.4 Scaling `Nabs`

Vertical scaling of `Nabs` is achieved through feature selection and throttling the flows using lossy counting. `Nabs` can also be scaled horizontally by deploying it on a pool of machines. The challenge is dividing the flows such that we guarantee each flow is handled by one and only one machine and all flows are handled by the pool. A simple hash-based sampling can effectively schedule the flows among the machines. Suppose we hash the `flow-id` and each machine in the flow is responsible for a non-overlapping range in the domain of the hash function, the scheduling works as follows. For each packet, machines compute the hash of `flow-id`. The packet is processed by the machine that is responsible for the range in which the hash value fall while the other machines discard the packet. This strategy ensures all flows are processed by the pool and only one machine process any given flow.

7 Related Work

Over the past few years significant research has been done to characterize network flows. Network traffic characteristics of various applications such as, web, email, and multimedia streaming, have been studied to support emerging network traffic trends for improving the underlying protocols. For this purpose, researchers have looked into various characteristics of network traffic such as, size of packets, inter-packet timings, round trip times, and transmission protocols. Network security community have borrowed some of these ideas and extended some others to improve the security of networks by identifying malicious network flows, applications, or hosts. In this section we briefly discuss prior work on network traffic characterization related to network security and refer the readers to [5] for a survey on traffic characterization in general.

In [13] a method is presented that uses the neural network to learn the signature of common network services and then monitor the network to detect flows that deviate from the norm. The authors use the total number of bytes transferred as a single feature to distinguish between Telnet and FTP traffic on networks. In [4], authors propose a method to identify well-known applications being tunneled through unconventional ports. The proposed method uses a decision tree algorithm to learn the statistical properties of various applications. The model learned is then

used to characterize the application types of network flows. Thanks to weak port bindings, port numbers are not considered in learning the model of both of these works.

A related problem to the above is tracing connection chains over multiple networks. Attackers often obscure their identity and location by forming a connection chain by logging into a set of compromised systems before attacking a target. Tracing the attack from the victim takes us only to the last link in the chain but not to the location of the attacker. In [12, 15], methods are proposed to trace intruders who obscure their identity by logging through a chain of multiple machines, known as stepping-stones. The method proposed in [12] creates "thumb-prints" of connections using packet content which can be compared to determine whether two connections contain the same content and are therefore likely to be part of the same connection chain. However, the method fails when the connections are encrypted. To address the problem [15] proposes an algorithm that doesn't rely on traffic content, instead relies on packet sizes, packet intervals, etc. to detect stepping stones.

8 Conclusion and Future Work

In this paper we introduced a system that characterizes content types of flows using only the payload. We presented the design and implementation details of a system which can be used for this purposes. We proposed and analyzed two throttling mechanisms to scale the system for deployment on large networks. We also identified 6 statistical properties of payloads that can characterize content types effectively. The proposed system was then used to detect abuses of network resources on a live network over a period of two weeks. The system performed well in detecting many cases of abuses which were missed by the firewall and intrusion detection systems.

We are currently developing a query processor that could integrate both continuous and instantaneous queries seamlessly. We are also looking to improve the accuracy of the classifier on compressed contents. Finally, we plan to design an active abuse detection system that can automate the enforcement of use policies on large networks.

References

[1] libpcap packet capture library. http://www.tcpdump.org/.

[2] Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[3] James P. Early, Carla E. Brodley, and Catherine Rosenberg. Behavioral authentication of server flows. In *Nineteenth Annual Computer Security Applications Conference*, pages 46–55, Las Vegas, Nevada, USA, December 2003.

[4] Sunita Kode, Jiten Maheswary, Mukta Nandwani, and Shilpa Suresh. Traffic characterization for heterogeneous applications. In *Technical Report*, May 2001.

[5] Gurmeet Singh Manku and Rajeev Motwani. Approximate frequency counts over data streams. In *Proceedings of the 28th VLDB Conference*, Hong Kong, China, November 2002.

[6] J. Mendel. Tutorial on higher-order statistics (spectra) in signal processing and system theory: Theoretical results and some applications. *IEEE Proceedings*, 79:278–305, March 1991.

[7] S. Muthukrishnan. Data streams: Algorithms and applications. http://www.cs.rutgers.edu/ muthu/ccmfun.pdf.

[8] R. Radhakrishnan, K. Shanmugasundaram, and N. Memon. Data masking: A secure-covert channel paradigm. St. Thomas, US Virgin Islands, 2002. IEEE Multimedia Signal Processing.

[9] P. Somol, P. Pudil, J. Novovicov, and P. Paclk. Adaptive floating search methods in feature selection. *Pattern Recognition Letters*, 20:1157–1163, 1999.

[10] S. Staniford-Chen and L.T. Heberlein. Holding intruders accountable on the internet. Oakland, 1995. Proceedings of the 1995 IEEE Symposium on Security and Privacy.

[11] K. M. C. Tan and B. S. Collie. Detection and classification of TCP/IP network services. In *Thirteenth Annual Computer Security Applications Conference*, pages 99–107, San Diego, California, USA, December 1997.

[12] V. Vapnik. The nature of statistical learning theory. *Springer-Verlag, New York*, 1995.

[13] Yin Zhang and Vern Paxson. Detecting stepping stones. In *Proceedings of the 9th USENIX Security Symposium*, Denver, Colorado, USA, August 2000.