2009 29th IEEE International Conference on Distributed Computing Systems

High-speed Flow Nature Identification

Amir R. Khakpour Alex X. Liu
Department of Computer Science and Engineering
Michigan State University
East Lansing, 48824 Michigan
{*khakpour, alexliu*}*@cse.msu.edu*

Abstract

This paper concerns the fundamental problem of identifying the content nature of a flow, namely text, binary, or encrypted, for the first time. We propose Iustitia, a tool for identifying flow nature on the fly. The key observation behind Iustitia is that text flows have the lowest entropy and encrypted flows have the highest entropy, while the entropy of binary flows stands in between. The basic idea of Iustitia is to classify flows using machine learning techniques where a feature is the entropy of every certain number of consecutive bytes. The key features of Iustitia are high speed (10% *of average packet inter-arrival time) and high accuracy (*86%*).*

1. Introduction

1.1. Motivation

In this paper, we aim to identify the content nature of flows at high speed. We define the nature of a flow as *text*, *binary*, or *encrypted*. A text flow mainly consists of words of natural languages. Typical text flow content include HTML pages, email, chat, telnet, etc. A binary flow mainly consists of binary content, such as executable code, multimedia files, etc. An encrypted flow mostly consists of traffic over SSL, encrypted files, etc. Fast identification of flow nature can enable a variety of applications in networking. Below we give a few examples.

Network monitoring and management. High-speed flow nature identification allows ISPs to prioritize their traffic for better quality of service. Considering an ISP serving a bank and a call center, among the traffic to/from the bank network, the ISP may give higher priority to the encrypted flows because they most likely carry banking transactions. Among the traffic to/from the call center, the ISP may give higher priority to the binary flows because they most likely carry voice data. High-speed flow nature identification also allows ISPs to gather various statistics on traffic passing through a network. Currently, up to 40% of Internet traffic belongs to unknown applications [1]. Identifying the nature of flows will help ISPs to understand better the type of traffic passing through their networks.

Forensics. High-speed flow nature identification allows ISPs to properly monitor and efficiently log their traffic for law enforcement purposes. For example, identifying binary flows may help copyright enforcement as they may carry copyrighted software and multimedia. As another example, identifying text flows may allow law enforcement to perform complex keyword searching for finding possible human communications on the fly.

Performance improvement for Intrusion Detection/Prevention Systems (IDS/IPS). With the rapid growth of Internet traffic and the increasing complexity of IDSes/IPSes, such security devices are becoming performance bottlenecks. When performing deep packet inspection on each packet cannot meet the performance demand of users, high-speed flow nature identification allows an IDS/IPS to apply binary related attack signatures on binary flows and text related attacks signatures on text flows, which is more efficient than applying all signatures on all flows.

Although identifying flow nature is a fundamental networking problem, to our best knowledge, it has not been investigated in prior literature. High-speed flow nature identification is a technically challenging problem. First, flow nature identification cannot rely on well-known port numbers. Any port number can be potentially customized for any application. Thus, flow nature identification has to rely on examining packet payload. Second, examination of packet payload at high speed is difficult for high bandwidth links.

1.2. Iustitia Architecture

In this paper, we propose Iustitia[1], a fast content-based flow classifier that classifies flows into text, binary, and encrypted. When a flow is started, Iustitia quickly identifies its nature based on a small number of buffered packet payload and forwards it to its corresponding buffer. The key observation behind Iustitia is that text flows have the lowest entropy and the encrypted flows have the highest entropy, while the entropy of binary flows stands in between. The basic idea of Iustitia is to classify flows using machine learning techniques where a feature is the entropy of every certain number of consecutive bytes. To address rigid time and space requirements in high speed routers, Iustitia leverages entropy estimation techniques and fast classification methods.

The architecture of Iustitia is shown in Figure 1. The classifier has a Classification Database (CDB) that contains flow IDs with their corresponding class label. Upon arrival of a packet, the classifier calculates the hash of the packet header to identify its flow ID. If the flow ID is found in CDB, the flow splitter forwards the packet to its corresponding output queue. Otherwise, the packet is buffered in its corresponding queue. When the buffer of a flow is full, a set of entropy-based classification features are extracted, where a feature

1. Iustitia (aka Lady Justice) is the Roman goddess of Justice, depicted and sculptured mostly blindfolded with a sword and measuring balances.

1063-6927/09 $25.00 © 2009 IEEE
DOI 10.1109/ICDCS.2009.34

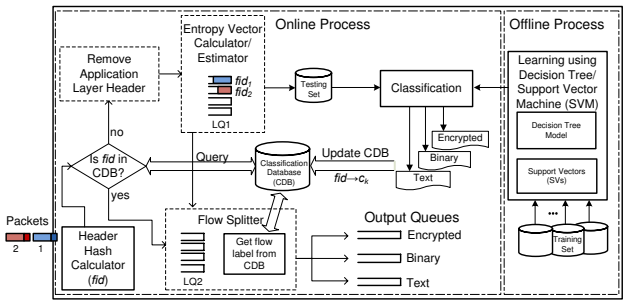


Figure 1. Iustitia Architecture

is the entropy of every certain number of consecutive bytes, and then fed into the classification module. The output of the classification module is a label of text, binary, or encrypted. Note that the classification module is trained offline using labeled data. When the nature of a flow is identified, the flow ID along with its corresponding label are stored in the classification database (CDB). Flows carried by TCP are removed from CDB when FIN or RST packets are seen. Flows in CDB are also removed when they are inactive for a certain amount of time.

1.3. Summery of Experimental Results

Our experimental results show that Iustitia can classify flows by their first 32 bytes of the data stream in about $300\mu s$ using 200 bytes of space per new flow with an average accuracy rate of 86%. The misclassification rates for text, binary, and encrypted flows are 4%, 12%, and 20%, respectively. With larger buffers, Iustitia can achieve an average accuracy rate of 90%. Based on our experiments on public traffic traces, on average, the delay caused by Iustitia is 10% of the average packet inter-arrival time; in more than 70% of the experimented flows, the delay caused by Iustitia is 5% of the average packet inter-arrival time.

The rest of the paper proceeds as follows. We first review related work in Section 2. We then define entropy vector and present how files can be classified by their entropy vector in Section 3. In Section 4, we adapt and optimize our file classification method for classifying flows. Finally, we give concluding remarks in Section 5.

2. Related Work

To our best knowledge, there is no prior work on methods for classifying flows as text, binary, and encrypted. Prior work on traffic classification for anomaly detection purposes mainly focuses on the inspection of packet headers and the detection of different network attacks using the statistical features of packet header fields [10], [11], [13]. The anomalies that can be detected and identified with such techniques are limited to attacks that can be detected based on packet headers, such as port scans, DOS, etc. However, these techniques are unable to discern attacks that cannot be detected solely based on packet headers such as malware, viruses, etc. Prior work on packet payload classification using machine learning techniques focuses on identifying what application the flow is associated with or whether a flow contains code. Signature-based approaches include the schemes for detecting well-known application protocols using packet size, timing characteristics, and packet payload [5], [6], [21]. For example, Early *et al.* built a decision tree classifier with n-grams of packets for detecting application protocols using average packet size and inter-arrival time along with TCP flags [6]. Bernaille *et al.* proposed an online flow classifier that uses first five packets of a flow to detect application protocols [4]. Machine learning techniques have also been used to infer application protocols from encrypted flows [20]. Lyda *et al.* used entropy analysis for detecting packed and encrypted malware on Portable Executable (PE) formatted files [15].

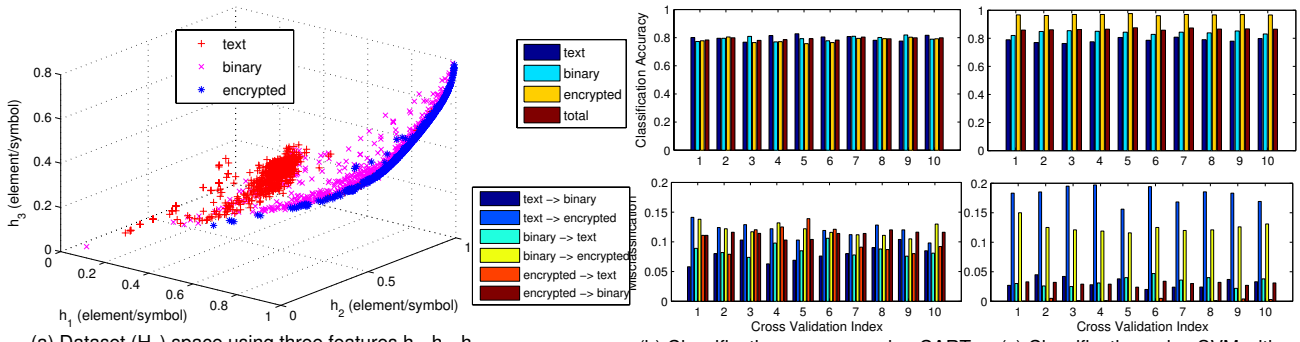
3. File Classification Using Entropy Vector

In this section, we first define the concept of file entropy vector. Second, we present two hypotheses, which will be used as the basis of Iustitia. Third, we show to what level of accuracy these hypotheses are true and how file classification can be useful for flow classification.

3.1. Entropy Vector

According to information theory, the entropy of a sequence of m elements over set $S = \{e_1, e_2, \ldots, e_n\}$, where each element is in S, is defined as $-\sum_{i=1}^{n} \frac{m_i}{m} \log(\frac{m_i}{m})$, where m_i is the frequency of element e_i. In this paper, we use normalized entropy where the base of logarithm is n and the unit of entropy is element per symbol. Note that we assume $0 \log 0 = 0$. The entropy of a sequence of elements is a measure of the diversity or randomness of the sequence. The minimum entropy value is 0 if all elements in the sequence are the same, and the maximum entropy value is 1 if all elements are repeated equally in the sequence. For simplicity, we use "entropy" to mean "normalized entropy" in the rest of the paper unless specified otherwise.

Given a file F, we can treat every byte as an element and F as a sequence of bytes, and henceforth calculate the entropy of the file over the set of all possible bytes. In general, we can treat every consecutive k bytes in file F as an element and calculate the entropy of the file over the set of all possible k bytes. For example, given $F = \langle a, b, c, d \rangle$, we can treat each consecutive 2 bytes as an element and the file as a sequence of 2 bytes $\langle ab, bc, cd \rangle$, and calculate the entropy of the file over the set of all possible 2^{16} two bytes. We use f_k to denote the set of all possible k bytes, and h_k to denote the entropy of the file F that is treated as a sequence of consecutive k bytes over set f_k. We calculate h_k in Formula (1). Note that $|f_k| = 2^{8k}$. We define an *entropy vector* for a given file F of m bytes, as a vector of elements where each element is in $\{h_1, \cdots, h_m\}$. We use H_F to denote the entropy vector for file F, and H_b to denote the entropy vector of the first b bytes of a file.



(a) Dataset (H_F) space using three features h_1, h_2, h_3 (b) Classification accuracy using CART (c) Classification using SVM with RBF kernel (C=1000, γ=50)

Figure 2. File entropy vector points in space and file classification accuracy using CART and SVM

	Decision Tree (CART)			
	Accuracy	Misclassification		
Total	79.19%	Text	Binary	Encrypted
Text File	79.94%	-	8.09%	11.97%
Binary File	79.34%	8.57%	-	12.05%
Encrypted File	78.25%	10.46%	11.29%	-
	SVM - RBF Kernel (γ=50,C=1000)			
	Accuracy	Misclassification		
Total	86.51%	Text	Binary	Encrypted
Text File	78.65%	-	3.18%	18.16%
Binary File	84.11%	3.35%	-	12.56%
Encrypted File	96.79%	0.20%	3.01%	-

Table 1. File classification using h_1 to h_{10}.

$$
\begin{aligned}
h_k &= -\sum_{i=1}^{|f_k|} \frac{m_{ik}}{m-k+1} \log_{|f_k|}\left(\frac{m_{ik}}{m-k+1}\right) \\
&= \frac{1}{m-k+1}\left[\sum_i m_{ik}\log(m-k+1) - \sum_i m_{ik}\log m_{ik}\right] \\
&= \log(m-k+1) - \frac{1}{m-k+1}\sum_i m_{ik}\log m_{ik} \qquad (1)
\end{aligned}
$$

In Formula (1), m_{ik} is the number of i-th element in f_k (i.e. $\sum_{i=1}^{|f_k|} m_{ik} = m-k+1$). In our flow classifier, given a file F of size m bytes, for every $1 \leqslant i \leqslant m$, we treat h_i as a *feature* of F. We call i the *feature width* of h_i.

3.2. Hypotheses and Validation

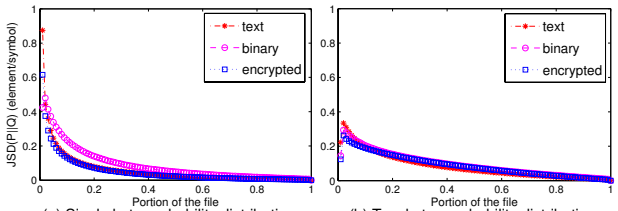
We make the following two hypotheses, which will serve as the basic of our flow classification scheme: (1) Using file entropy vectors, we can classify files into *text*, *encrypted*, and *binary* files. (2) The randomness of the beginning portion of a file represents the randomness of the entire file.

The first hypothesis is based on the observation that in comparison with binary files, text files have lower entropy as they tend to have repeated elements and encrypted files have higher entropy as their content are most random. The second hypothesis is based on the observation that the randomness of the first b bytes of a file is typically close to randomness of the whole file.

To validate these two hypotheses, we collected a pool of three classes of files: 52,273 binary files (including executables, JPG, GIF, AVI, MPG, PDF, ZIP files), 24,985 text files (including text documents, manuals, txt files, log files, htmls), and 13,656 encrypted files (generated using PGP, AES, DES, etc). We validated hypothesis 1 using 10 times cross-validation over these files. Each cross-validation uses 6000 files equally drawn from each class. For each file, we calculated an entropy vector of size 10. The feature space of the first three features h_1, h_2, h_3 are shown in Figure 2(a). As expected, most data points for text files have the lowest entropy values, most data points for encrypted files have the highest entropy values, and most data points for binary files stand in between. We then apply two classification methods, decision tree (CART) [9] and Support Vector Machines (SVM) [19] to each dataset. The results are shown in Figure 2(b) and 2(c) and Table 1.

Figure 2(b) indicates that by using decision trees, we can have up to 79% classification accuracy, where the prediction accuracy for each class is almost the same as the total accuracy (*i.e.*, the accuracy for all classes). Table 1 illustrates the average misclassification rate for each class trivially changes around 10%.

We also experimented with SVMs, which are commonly used in machine learning for complex feature spaces. After model selection, we achieved best classification accuracy rate on SVM with Radial Basis Function (RBF) kernel by $\gamma = 50$ and $C = 1000$ parameters. For multi-class classification, we also use DAGSVM [16], which is the fastest among other multi-class voting methods [7]. Using this classification model, the total accuracy rate is improved up to 86%. The results show that even though the classification accuracy for text data does not improve much, the accuracy for encrypted data and binary data has significant improvement. The accuracy for encrypted data is improved from 77% to 96% and that for binary data is improved from 80% to 85%. Because of the nonlinear nature of the RBF kernel, the misclassification pattern in RBF kernel is quite different from that in the decision tree approach. For instance, the misclassification rate for text data points that are classified as encrypted is increased up to 8%, and the misclassification rate for encrypted data points that are classified as text is decreased up to 10%. To validate hypothesis 2, we use Jensen-Shannon divergence



(a) Single byte probability distribution distance (b) Two bytes probability distribution distance

Figure 3. The JSD distance of element frequency distribution of the first b bytes of the file and the whole file content

(JSD) [14] (also known as total divergence to the average) measure which is a popular method to calculate the distance between two probability distributions. In fact, JSD is the average of Kullback-Leibler (KL) distances of two distributions to their average distribution. JSD is smooth, bounded [0,1], and satisfies the symmetry condition. Let P and Q be two probability distributions. The KL distance (also known as relative entropy) between P and Q is defined as $KLD(P\|Q) = \sum_i p_i \log \frac{p_i}{q_i}$. Let M be the average probability distribution ($M = \frac{P+Q}{2}$). The Jensen-Shannon divergence denoted by $JSD(P\|Q)$ is defined as follows. Note that $JSD(P\|Q) = 0$ if and only if $P = Q$.

$$JSD(P\|Q) = \frac{1}{2}KLD(P\|M) + \frac{1}{2}KLD(Q\|M)$$

$$= \frac{1}{2}(\sum_i p_i \log \frac{p_i}{m_i} + \sum_i q_i \log \frac{q_i}{m_i}) = H(M) - \frac{1}{2}H(P) - \frac{1}{2}H(Q) \quad (2)$$

In hypothesis 2, P is the byte probability distribution of the first b bytes of a file and Q is the byte probability distribution for the entire file. We calculate $JSD(P\|Q)$ for single byte element set (f_1), two bytes element set (f_2), and three bytes element set (f_3). Figure 3 illustrates the average of JSD distance over 1000 files for each class. Since file sizes differ, for each file, b is chosen as a portion of the total file. Figure 3(a) depicts that for all classes of files, the probability distribution of f_1 for the first 20% of the file can represent the entire file with more than 86% of similarity. Figure 3(b) shows that the similarity rate for probability distribution of f_2 is 70% and as it is shown in [8] JSD similarity is 67% for f_3.

4. Flow Classification Using Entropy Vector

Treating a data flow as a sequence of bytes, we can classify flows using our hypotheses in Section 3. As there are no real-world packet traces with payload to be investigated for our experiments because of legal and privacy reasons, we experiment with real-world packet traces with payload randomly chosen from our collected files. To design a flow classifier, there are some parameters that need to be carefully chosen for better accuracy and efficiency. To accelerate the entropy calculation/estimation process and reduce the size of the classification model, we first use feature selection methods to shrink the feature space without considerable accuracy decrease. We then choose buffer size b such that it is small enough to avoid long delays and large space usage, and big enough to keep the accuracy rate high. We further address the time and space requirements using entropy vector estimation.

	Decision Tree (CART)	
	$H_F = \langle h_1 \dots h_{10} \rangle$	$H_F = \langle h_1, h_3, h_4, h_{10} \ / \ h_5 \rangle$
Total	79.19%	79.20 / 78.61%
Text File	79.94%	79.66 / 79.35%
Binary File	79.34%	79.27 / 78.49%
Encrypted File	78.25%	78.66 / 77.99%
	SVM - RBF Kernel (γ=50,C=1000)	
	$H_F = \langle h_1 \dots h_{10} \rangle$	$H_F = \langle h_1, h_2, h_3, h_9 \ / \ h_5 \rangle$
Total	86.51%	86.08% / 85.41%
Text File	78.65%	78.91% / 78.91%
Binary File	84.11%	83.40% / 82.28%
Encrypted File	96.79%	95.94% / 95.04%

Table 2. Classification accuracy after feature selection

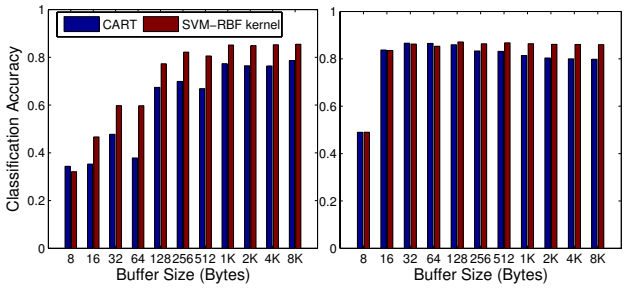
4.1. Feature Selection

There are many feature selection methods that can reduce the number of features and feature space. For the decision tree approach, the higher a feature is in a tree, more effective a feature is in the classification model. We use a voting scheme for the feature selection on 10 decision trees of 10 cross-validation. In order to find distinctive features, we prune the trees until we reach the threshold of 2% decrease in accuracy. Then, the features that are more frequently used in the pruned trees are chosen as the selected features. The feature set denoted by ϕ_{CART} and calculated by this method on our dataset is $\phi_{CART} = \{h_1, h_3, h_4, h_{10}\}$. For SVM classification, we use the Sequential Forward Searching (SFS) algorithm [17] for feature selection. Let n be the total number of features, and n' be the number of selected features. In SFS, we start with an empty set of features. On each iteration, we add one feature and evaluate the classification accuracy rate using SVM. Then, we choose the feature that has the maximum accuracy rate among all features. The search terminates until we select n' features. As we apply this algorithm on all cross-validation sets, we use a voting mechanism to choose the best features. The selected feature set using this algorithm is $\phi_{SVM} = \{h_1, h_2, h_3, h_9\}$. In flow classification, we prefer features h_k with lower k value because in calculating an entropy vector, as k increases, the number of elements in f_k increases exponentially, which may cause the classification algorithm to use more memory. Factoring in this preference, the feature sets that we choose are $\phi'_{CART} = \{h_1, h_3, h_4, h_5\}$, and $\phi'_{SVM} = \{h_1, h_2, h_3, h_5\}$.

Table 2 shows that after feature selection classification accuracy rates slightly change where in some cases even increase. It also shows that the feature sets that we chose after factoring in feature preferences have little impact on classification accuracy compared to the original feature sets.

4.2. Choosing Buffer Size

Next, we discuss how to determine the size of the buffer used to store packet payload for calculating entropy vectors.



(a) Classification acuracy by training using entire file (b) Classification acuracy by training using fisrt b byte of file

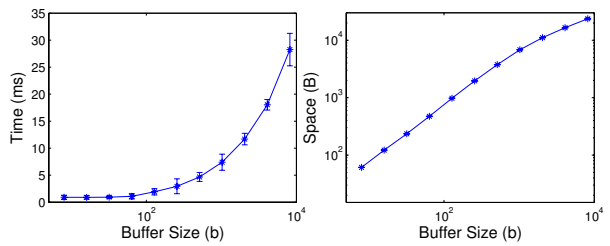
Figure 4. Classification accuracy based on b

To achieve high time and space efficiency, this buffer needs to be small. To achieve high classification accuracy, this buffer needs to be large enough. Finding a good tradeoff for the buffer size is an important issue. The optimal value of b depends on how the classifier was trained. If we train the classifier using the entire content of every training file and classify flows using their first b bytes, Figure 4(a) shows that we can achieve an accuracy rate of 86% using a buffer of 1K bytes with the SVM classification model. If we train the classifier using the the first b bytes of every training file and classify flows using their first b bytes, Figure 4(b) shows that we can achieve an accuracy rate of 86% using a buffer of 32 bytes with both the SVM and the decision tree classification models. Note that for SVM, we use the same classification model that was used for the file classification.

To compare two buffer sizes based on the two training methods in terms of time and space, we implement our classifier using C++ on a Linux workstation with AMD 64 Athlon Dual core and 2.5 GB of RAM. The results shown in Figure 5 indicate that the second training method is preferred over the first training method. According to Figure 5(a), the entropy vector calculation time using the second training method with $b = 32$ is almost 10 times less than the first training method with $b = 1024$. Furthermore, according to Figure 5(b), the space per flow using the second training method is 30 times less than first training method. Note that both time and space curves increase linearly as buffer size increases.

4.3. Handling Application Layer Headers

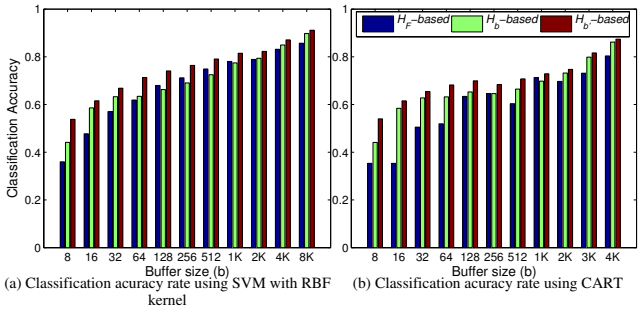
Many application protocols have headers. For instance, a picture in an html page is fetched from a web server in a separate flow with an HTTP header in text. Such a binary flow with application headers in text may be misclassified as our classification is based on the first b bytes of the flow.



(a) Entropy vector calculation time (b) Entropy vector calculation space

Figure 5. Entropy vector calculation

For headers of well-known application protocols, such as HTTP, SMTP, IMAP, and POP, as they have a known format, our classifier strips them off using signature based header detection techniques [5], [21] and uses only application payload for classification purposes. However, based on Internet flow statistics [1], known application flows and encrypted flows constitute around 50% and 10% of the Internet traffic, respectively. Thus, almost 40% of traffic flows are unknown. This includes the flows that have no application headers (such as FTP file transfer flows and most P2P network file transfer flows) and the flows that have unknown application headers. As the size of an application header varies for different applications, we use a threshold T as a maximum number of header bytes. In other words, to calculate an entropy vector, we treat the $(T + 1)$-th byte in a flow as the beginning of the flow. To examine classification accuracy using threshold T, for each experimented flow, we randomly choose an application header length Y ($Y \leqslant T$). We then treat the $(T-Y+1)$-th byte as the beginning of the flow. We now have three methods to train our classifier:(1)H_F-based method that uses entire file content, (2) H_b-based method that uses the first b bytes of file content, (3) $H_{b'}$-based method that uses b consecutive bytes starting at a random location ranging from the first byte to the $(T + 1)$-th byte. Recall that H_F denotes the entropy vector for file F and H_b denotes the entropy vector of the first b bytes of a file. Figure 6 compares the accuracy of different classification models using different buffer sizes. For both classification models of SVM and CART, Figures 6(a) and (b) show that classification accuracy does not significantly differ when we use the above three training methods. This is expected as we have already shown that the characters of a flow probability distribution does not change significantly over the flow content. The figure also shows that using larger buffer sizes may increase classification accuracy for both classification models. Comparing the two classification models, SVM with the RBF kernel achieves up to 10% better accuracy than CART for most buffer sizes. Our experimental results show that with unknown application headers removed, our classifier achieves a classification accuracy of 80% with a buffer of 1024 bytes.



(a) Classification acuracy rate using SVM with RBF kernel (b) Classification acuracy rate using CART

Figure 6. Classification accuracy by different type of training sets using SVM and CART

4.4. Entropy Vector Estimation

4.4.1. Overview. Calculating exact entropy vectors for each flow can be costly in time and space when we use a buffer of 1K bytes. In Iustitia, we use a (δ, ϵ)-approximation algorithm to estimate $S_k \equiv \sum_i m_{ik} \log m_{ik}$ in Formula (1). This algorithm is proposed for estimating the entropy of data streams by Lall et al. in [12]. The basic idea is to use frequency moments estimation techniques [3]. The estimation algorithm has relative error of at most ϵ with probability at least $1 - \delta$. Thus, given X whose estimated value is $\tilde{X}$, we have $Pr(|X - \tilde{X}| \leqslant X\epsilon) \geqslant 1 - \delta$. Note that the entropy estimation algorithm is based on fundamental assumption of $|f_i| \gg b$. This implies that we cannot use the estimation algorithm for h_1 because $|f_1| = 256$. However, the rest of the features use the estimation algorithm as $|f_i| = 256^i$.

In the (δ, ϵ)-approximation algorithm, the number of counters required for entropy estimation is significantly less than the number of counters required for accurate entropy calculation because the number of elements in f_i increases exponentially. The number of counters for feature ϕ_i is $g \times z_i$. Note that g and z_i are calculated based on δ and ϵ as follows: $z_i = \lceil 32 \log_{|f_i|} b/\epsilon^2 \rceil$, and $g = 2\log_2(1/\delta)$.

In this paper, as the main objective is to obtain higher classification accuracy with smaller number of counters, we examine classification accuracy for each possible value of ϵ and δ. To save space in entropy estimation, we need to ensure that the number of counters is less than the number of counters in actual entropy vector calculation. Theoretically, we assign one counter for each element of a feature, thus the total number of counters is $(\sum_{\forall \phi_i \neq h_1} |f_i|)$. However, in practice, most of the counters are 0 as $|f_i| \gg b$. If we assume α as the total number of counters in exact calculation, we can calculate the lower bound for ϵ and δ as follows:

$$\sum_{\forall \phi_i \neq h_1} g \times z_i = \sum_{\forall \phi_i \neq h_1} \frac{32 \log_{|f_i|} b \cdot 2\log_2(1/\delta)}{\epsilon^2} < \alpha \quad (3)$$

Replacing $|f_i|$ by 2^{8i} will give us

$$\sum_{\forall \phi_i \neq h_1} \frac{8 \log_2 b \cdot \log_2(1/\delta)}{\epsilon^2 i} < \alpha \Rightarrow \epsilon > \sqrt{K_\phi \frac{\log_2 b}{\alpha} \log_2(1/\delta)} \quad (4)$$

where K_ϕ is the feature set coefficient and defined as $K_\phi = 8 \sum_{\forall \phi_i \neq h_1} \frac{1}{i}$. For our feature sets, $K_{\phi_{SVM}} = 8.26$, $K_{\phi_{CART}} = 6.26$. Also, for our dataset with $b = 1024$, we estimate $\alpha \simeq 1911$. Thus, for our dataset formula (4) is reduced to $\epsilon > 0.18\sqrt{\log_2(1/\delta)}$.

For each flow, when its buffer becomes full or the buffer stops receiving packets for a certain period of time, for each feature $\phi_k (k > 1)$, we estimate the value of h_k in the flow's entropy vector as follows. First, we randomly choose $g \times z$ locations in the buffer. Second, for each of the $g \times z$ locations, we pick k consecutive bytes in the buffer as an element, then scan the buffer starting from the location to the end and count the number of k consecutive bytes that are equal to the element. Now, we have a total of $g \times z$ counters. Third, we group these counters into g groups where each group has z counters. Fourth, for each counter c, we calculate $b(c \log c - (c-1)\log(c-1)))$ as the unbiased estimator for S_k. Fifth, for each group, we calculate the average of the z unbiased estimators. Sixth, we calculate the median of the g average values as the estimated value of S_k. Finally, we use Formula (1) to calculate the estimated value of h_k. The detailed pseudocode of the above entropy vector estimation algorithm is available in the technical report version of this paper [8].

4.4.2. Identifying ϵ and δ. To identify the best values for ϵ and δ, we calculate the entropy vector based on different values for ϵ and δ and examine the classification accuracy for each combination of ϵ and δ. As shown in Figure 2(a), the data points for different classes are close to each other in space, and they are not well segregated. This implies that the (δ, ϵ)-estimated entropy vector data points induce more overlaps rather than exact entropy vector data points. Thus, we expect the misclassification rate to increase. Based on the experimental results using the (δ, ϵ)-approximation algorithm shown in Figure 7, we make the following observations: (1) Using SVM with $H_{b'}$-based training method where $b' = 1024$, we have a classification accuracy of 81.3%, and the optimal values of ϵ and δ are 0.25 and 0.75, respectively. However, to improve classification accuracy, we reapply the model selection on the new dataset, and obtain a classification accuracy of 83% using $\gamma = 10$ and $C = 1000$. Figure 7(i) shows the classification accuracy rate for different classes using different ϵ and δ values classified by SVM with new model parameters. (2) Using CART with $H_{b'}$-based training method where $b' = 1024$, we have a classification accuracy of 76.03%, and the optimal values of ϵ and δ are 0.5 and 0.1, respectively. Figure 7(ii) shows the classification accuracy for different classes using different ϵ and δ values. (3) The entropy vector estimation algorithm is not effective for small buffers of size such as 32 bytes.

4.4.3. Time and Space Analysis. Table 3 compares the time and space requirement for calculating and estimating an entropy vector. Comparing the entropy vector exact calculation and estimation, the estimation process requires almost three times less memory, yet it takes three times more time to be executed. In fact, the buffer size is not big enough for this algorithm to be efficient on time constraints; however, it reduces the required space significantly.

		Calculation		Estimation	
		Time	Space	Time	Space
b=1024B	SVM	5428μs	5.1KB	16421μs	1.6KB
	CART	5658μs	5.3KB	18751μs	1.85KB
b=32B	SVM	326μs	195B	-	-
	CART	276μs	192B		

Table 3. Comparing the required time and space for (b=1024B) and (b=32B)

4.5. Classifier Buffering Delay Analysis

As shown in Figure 1, the incoming packets need to be buffered and once the flow buffer is full with the required

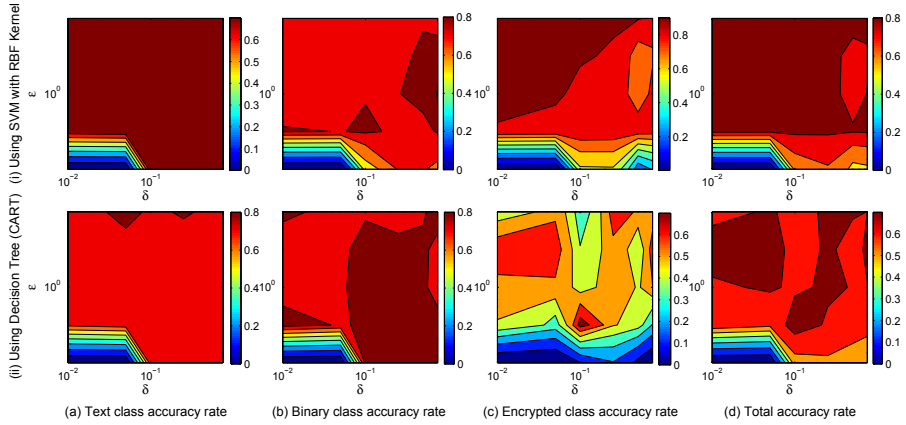


Figure 7. Classification accuracy based on different g and z values

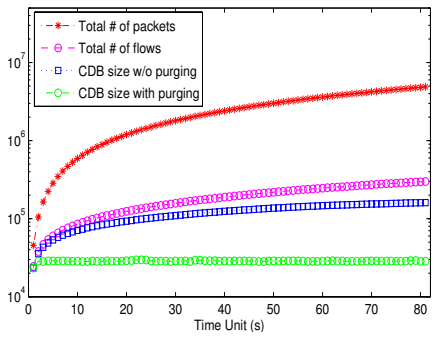


Figure 8. Comparing CDB size with total number of flows and total number of packets

b bytes, the buffer is sent to the entropy vector calculator/estimator unit for classification. In some applications, this part can be embedded into the application layer. In such cases, the flow classification overhead will be reduced to entropy calculation/estimation. Table 3 shows the classification calculation time (delay). However, if we want to consider the flow classifier as an independent component in a network, we need to evaluate the delay of the flow buffering. For this purpose, we experimented with a gigabit gateway trace from the UMASS Trace Repository [2], [18] that has 11,976,410 packet records where 41.16 % of them are the UDP or TCP data packets. The packet rate in this trace is 146,714.38 packets per second. This trace contains 299,564 data flows.

We experimented with four buffer sizes: 32, 1024, 1500, 2000. We choose $b = 32$ and $b = 1024$ for systems that deal with flows with no application header, and $T+b' = 1500$ and $T+b' = 2000$ for systems that are required to cut the first T bytes of the flow to throw out the possible application header (T varies from 0 to 1970 based on the required classification accuracy and the classification model).

The total delay in the buffering stage τ can be calculated as the summation of the header hash calculation time (τ_{hash}), the CDB search time ($\tau_{CDBsearch}$), and the time period that requires for the buffer to be filled (τ_b).

We use SHA-1 to create 160 bit hash result for each flow $\mathbb{F}_i$. The hash calculation time is around $18\mu s$ which is trivial comparing to CDB search time and inter-arrival time. However, in order to have a reasonable CDB search time, we need to purge the obsolete flows from CDB, meaning that we need to make sure that the CDB size tends to be the number of concurrent (actual) flows. Therefore, once packets with RST or FIN flags are received, their corresponding flow record in CDB will be removed. Figure 8 illustrates that up to 46% of the flows are removed. However, some applications do not close the TCP sockets properly and also packets in UDP flows do not have FIN or RST flags. Hence, we use the packet arrival time to remove the obsolete flows. We assume that a flow $\mathbb{F}_i$ is obsolete if the following condition holds: $t_{current} - t_{\mathbb{F}_i} \geqslant n\lambda_{\mathbb{F}'_i}$, where $t_{current}$ is the current time, $t_{\mathbb{F}_i}$ is the arrival time of the last packet of the flow $\mathbb{F}_i$, n is an arbitrary coefficient, and $\lambda_{\mathbb{F}'_i}$ is the inter-arrival time from the last two packets for the flow $\mathbb{F}_i$. Note that if only one packet of a flow is received, we use default $\lambda = 0.5s$. This implies that we need to keep $\lambda_{\mathbb{F}'_i}$ in CDB for each flow. However, we can save memory and use a constant λ value for all flows. The cumulative probability (CDF) of $\lambda'_{\mathbb{F}}$ for our trace is shown in Figure 9(b).

The parameter n should be determined based on the given time and space requirement. In terms of space, we need to consider that each record in the CDB is 194 bits long including 160 bits for the SHA-1 hash result, 32 bits for $\lambda'_{\mathbb{F}_i}$, and 2 bits for the flow class label. Yet, small n may lead to unnecessary flow classification for the flows that have been classified before but have been removed from CDB. Thus, as flow classification required space (Table 3) could be noticeably high comparing to 192 bits, we are interested in n values that are large enough to avoid reclassification of the same flow. Our experimental results show that $n = 4$ is an optimal value for the given traffic trace. We also trigger the purging threads once the number of flows is increased by 5,000 flows. This value may vary based on the link rate and available resources. Figure 8 indicates that the CDB size after pruning is fairly constant on 29,713 flows over time.

Another important factor in delay analysis is the required time period for the buffer to be filled. This time is calculated as $\tau_b = \sum_{j=1}^{c} \lambda_j$ where c is the number of required packets to fill the buffer. Considering fixed buffer size b, the number

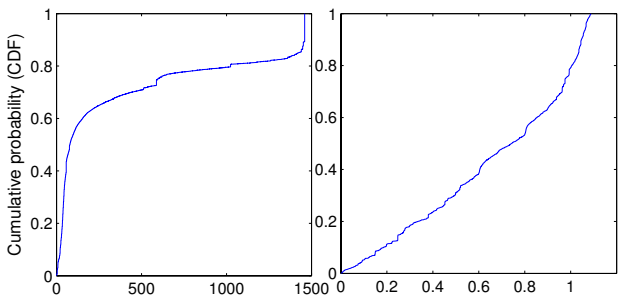
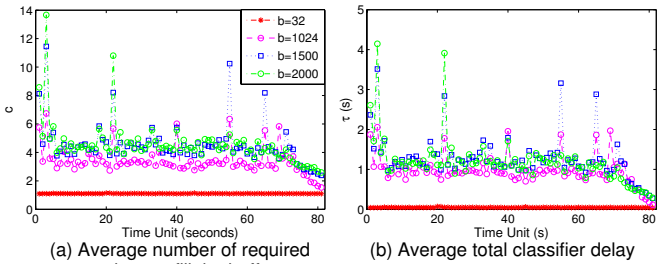


Figure 9. Cumulative probability distribution of packet size and inter-arrival time



(a) Average number of required packets to fill the buffer

(b) Average total classifier delay

Figure 10. Classification delay

of packets has inversely proportional to the packet payload size. The cumulative distribution of the packet payload size in the given packet trace shown in Figure 9(a) follows bimodal packet size probability distribution, where up to 20% of the packets have payload size of 1480 and more than 50% have payload size of less than 140 bytes. This implies that the average number of required packets to fill the buffer c is equal to 1 for b=32 and 3 to 5 for higher buffer sizes up to 2000 bytes as shown in Figure 10(a). Thus, for b=32 we have almost no inter-arrival time, whereas for higher buffer sizes τ_b varies from 2 to 4 inter-arrival time in average. As shown in Figure 10(b), the total delay of the classifier τ is dominated by τ_b where the total delay is $50ms$ for small buffer sizes and varies around $1s$ for big buffer sizes.

4.6. Discussion

Network tunneling adds more complication for flow classification. A tunnel may contain multiple flows with different natures. If the tunnel is encrypted, we classify the tunnel as an encrypted flow. If the tunnel is not encrypted, we should distinguish every flow inside the tunnel and classify them separately.

In case Iustitia is used for security applications, an attacker may defraud Iustitia by adding some deceiving padding in the beginning of a flow to cause misclassification. For instance, if we prioritize different output buffers such that binary flows have the highest priority and encrypted flows have the lowest priority, an attacker may put some encrypted-like padding to the beginning of a flow and send it over to bypass complex signature matching. To deal with this problem, one solution is to randomly skip the first T bytes in a flow and then begin to buffer packet payload for flow classification. An alternative solution is to periodically delete the CDB record of a flow that has started for a certain amount of time, which will cause the flow to be reclassified.

5. Conclusion

We make two major contributions in this paper. First, we propose the first method for classifying flows into text, binary, and encrypted at high-speed with a small amount of memory. Our method is mainly based on machine learning and information theory techniques. Second, we implemented our method and performed extensive experiments. Our experimental results show that our method achieves high speed (10% of average packet inter-arrival time) and high accuracy (86%).

Acknowledgement

The work of Alex X. Liu is supported in part by the National Science Foundation under Grant No. CNS-0716407. The authors would like to thank Rong Jin and Hamed Valizadegan for discussion on machine learning methods, Kiran Misra for his help on information theory, and Ashwin Lall for his comments on entropy estimation.

References

[1] Internet2 netflow statistics, http://netflow.internet2.edu/.
[2] Umass Trace Repository, http://traces.cs.umass.edu/.
[3] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *STOC '96: Proc. of ACM symposium on theory of computing*, 1996.
[4] L. Bernaille, R. Teixeira, I. Akodkenou, A. Soule, and K. Salamatian. Traffic classification on the fly. *SIGCOMM Comput. Commun. Rev.*, 36:23–26, 2006.
[5] H. Dreger, A. Feldmann, M. Mai, V. Paxson, and R. Sommer. Dynamic application-layer protocol analysis for network intrusion detection. In USENIX Security Symp., 2006.
[6] J. P. Early, C. E. Brodley, and C. Rosenberg. Behavioral authentication of server flows. In *ACSAC '03: Proc. of the 19th Annual Computer Security Applications Conf.* , 2003.
[7] C.-W. Hsu and C.-J. Lin. A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, 13:415–425, 2002.
[8] A. R. Khakpour and A. X. Liu. Iustitia: An information theoretical approach to high-speed flow nature identification. Technical Report MSU-CSE-09-7, 2009.
[9] R. O. L. Breiman, J.H. Friedman and C. Stone. *Classification and Regression Trees.* Chapman & Hall, 1984.
[10] A. Lakhina, M. Crovella, and C. Diot. Diagnosing network-wide traffic anomalies. In *SIGCOMM '04*, 2004.
[11] A. Lakhina, M. Crovella, and C. Diot. Mining anomalies using traffic feature distributions. *SIGCOMM Comput. Commun. Rev.*, 35:217–228, 2005.
[12] A. Lall, V. Sekar, M. Ogihara, J. Xu, and H. Zhang. Data streaming algorithms for estimating entropy of network traffic. In *Proc. of SIGMETRICS '06*, pages 145–156, 2006.
[13] W. Lee and D. Xiang. Information-theoretic measures for anomaly detection. In *Proc. of the IEEE S&P*, 2001.
[14] J. Lin. Divergence measures based on the shannon entropy. *IEEE Tran. on Information Theory*, 37:145–151, 1991.
[15] R. Lyda and J. Hamrock. Using entropy analysis to find encrypted and packed malware. *Proc. of IEEE S&P*, 2007.
[16] J. C. Platt, N. Cristianini, and J. Shawe-taylor. Large margin DAGs for multiclass classification. In *Advances in Neural Information Processing Systems*. MIT Press, 2000.
[17] P. Somol, P. Pudil, J. Novovicova, and P. Paclik. Adaptive floating search methods in feature selection. *Pattern Recognition Letter*, 20:11–13, 1999.
[18] K. Suh, Y. Guo, J. Kurose, and D. Towsley. Locating network monitors: complexity, heuristics, and coverage. *in Proc. INFOCOM 2005*, 1:351–361, 2005.
[19] V. N. Vapnik. *The nature of statistical learning theory*. Springer-Verlag New York, Inc., 1995.
[20] C. V. Wright, F. Monrose, and G. M. Masson. On inferring application protocol behaviors in encrypted network traffic. *Journal of Machine Learning Research*, z:2745–2769, 2006.
[21] Y. Zhang and V. Paxson. Detecting backdoors. In *Proc. of USENIX Security*, 2000.