CUTE: traffic Classification Using TErms

Soheil Hassas Yeganeh
University of Toronto
soheil@cs.toronto.edu

Milad Eftekhar
University of Toronto
milad@cs.toronto.edu

Yashar Ganjali
University of Toronto
yganjali@cs.toronto.edu

Ram Keralapura
Narus Inc., CA, US
rkeralapura@narus.com

Antonio Nucci
Narus Inc., CA, US
anucci@narus.com

***Abstract*—Among different traffic classification approaches, Deep Packet Inspection (DPI) methods are considered as the most accurate. These methods, however, have two drawbacks: (*i*) they are not efficient since they use complex regular expressions as protocol signatures, and (*ii*) they require manual intervention to generate and maintain signatures, partly due to the signature complexity.**

In this paper, we present CUTE, an automatic traffic classification method, which relies on sets of weighted terms as protocol signatures. The key idea behind CUTE is an observation that, given appropriate weights, the occurrence of a specific term is more important than the relative location of terms in a flow. This observation is based on experimental evaluations as well as theoretical analysis, and leads to several key advantages over previous classification techniques: (*i*) CUTE is extremely faster than other classification schemes since matching flows with weighed terms is significantly faster than matching regular expressions; (*ii*) CUTE can classify network traffic using only the first few bytes of the flows in most cases; and (*iii*) Unlike most existing classification techniques, CUTE can be used to classify partial (or even slightly modified) flows. Even though CUTE replaces complex regular expressions with a set of simple terms, using theoretical analysis and experimental evaluations (based on two large packet traces from tier-one ISPs), we show that its accuracy is as good as or better than existing complex classification schemes, *i.e.* CUTE achieves precision and recall rates of more than 90%. Additionally, CUTE can successfully classify more than half of flows that other DPI methods fail to classify.

***Index Terms*—Traffic Classification, Deep Packet Inspection, Automatic Signature Generation**

I. Introduction

Traffic classification is an important element in computer networks with many applications including traffic prioritization, bandwidth management, and intrusion detection. There are different categories of traffic classification methods. *Port-based* methods are the oldest, the simplest, and the most efficient ones which classify flows based on standard port numbers. Despite being efficient, port-based methods have a very limited scope since they fail to detect protocols that are carried over non-standard ports. For instance, many peer-to-peer protocols use random ports for communication, and therefore are not detectable by port-based classification methods.

Feature-based methods, on the other hand, use selected flow features to classify network flows. Although these methods can detect several protocol families (*e.g.,* email, web, peer-to-peer, etc.), they have poor performance in identifying the exact protocol which might be essential for some applications. Moreover, they are prone to be over-trained, and are very sensitive to the learning data set [19].

Deep Packet Inspection (DPI) methods are the most accurate as they inspect the payload (or content) of network flows. These methods utilize a set of *protocol signatures* for traffic classification. A signature is a set of regular expressions that match flow payloads for a specific protocol, as well as additional flow information such as port numbers.

Despite their high accuracy levels, DPI methods suffer from two major drawbacks: (*i*) Signature generation is very challenging, and usually requires laborious manual effort. Thus, adapting signatures to temporal changes for existing protocols is extremely difficult. (*ii*) Even for methods which automatically generate signatures, the output is in the form of very complex regular expressions. Matching flow payloads against these signatures either takes a very long time to complete, or requires prohibitively large space, making it difficult to work in realtime, and at extremely high line rates.

To tackle these issues, we present CUTE: an automatic traffic classification system with extremely simple signatures. Instead of using complex regular expressions as protocol signatures, CUTE relies on weighted term sets. These signatures are easy to extract automatically, simple to match against flows in realtime, and result in very high accuracy levels. A set of weighted terms can be considered as a relaxed version of a regular expression, which implies that the *occurrence* of terms is more important than the *relative order* of terms in flow payloads. Needless to say, such a relaxation might introduce classification errors, specially when protocols share many common terms. However, in our experiments (with large packet traces collected from tier-1 ISPs) we observe that such scenarios are very unlikely, and therefore, we conclude that weighted term sets have high accuracy for traffic classification.

CUTE automatically extracts weighted terms for various protocols from a set of training flows. Based on these weighted terms, CUTE estimates the similarity of a flow to each protocol, and classifies the flow accordingly. CUTE is very effective even when we can monitor only the first few bytes of flows, or when we encounter partial/incomplete flows with asymmetric routing. Our experiments show that CUTE can reach a precision and recall rate of about 90% even if it uses the first 100 bytes of each flow. We note that incomplete flows might be common in real-world network traces.

To show the effectiveness of CUTE, we comprehensively analyzed two large datasets captured from two different tier-1 ISPs within two different continents. Our results show that CUTE achieves very high precision and recall rates ($> 90\%$) for almost all the protocols even when we use only 5% of the most important terms that CUTE extracts from the training flows. These rates are as good as or better than rates achieved by other automatic signature generation methods [18] (that usually focus on a limited set of flows such as

978-1-4673-1544-9/12/$31.00 ©2012 IEEE

file-sharing applications rather). We note that increasing the percentage of terms that are used by CUTE will result in better accuracy at the cost of reducing efficiency. This trade-off can be very useful for network operators that can find the right balance of accuracy and efficiency depending on their network conditions and needs. Interestingly, CUTE labels almost 60% of incomplete flows, which are classified as *unknown* flows by manually (and carefully) created protocol signatures in other DPI methods.

Our contributions are threefold: First, we propose a very efficient, and accurate automatic traffic classification method, which relies on weighted term matching. Second, using theoretical analysis as well as experimental evaluations, we show that occurrence of terms in network flows are more important than the relative order of the terms in traffic classification. This observation is the basis of CUTE, and explains why it can achieve high precision and recall rates with very simple signatures. We also show that CUTE can efficiently classify flows even when they have partial payloads. Finally, we show that existing methods for automatically generating signatures are biased toward the training dataset, and thus cannot robustly classify flows. We show that this problem can be solved by using partial matching techniques (such as CUTE), which can be robust to the training data set.

Next, we present CUTE in Section II), and evaluate it using real packet traces in Section III. We provide an abstract model for the automatic signature generation problem in Section IV, and theoretically show why CUTE works in Section V. Section VI presents related works, and is followed by a brief discussion in Section VII.

II. The Proposed Method

Manual signature authoring is a tedious and resource intensive task, that results in static protocol signatures which lose their accuracy over time as protocols evolve. In response, some automatic signature generation methods have been proposed, such as Longest Common Subsequence [21] and Laser [18], which generate regular expressions as protocol signatures. In other words, they find the best regular expressions that can classify network flows.

In this section, we present the proposed classification method, CUTE. CUTE uses terms for classification instead of using regular expressions. For example, existing DPI methods use "`GET.*HTTP`" as a signature for `HTTP`, but CUTE uses "`GET` or `HTTP`" as a signature for `HTTP`. In general, CUTE uses a highly relaxed regular expression for classifying flows. Note that such relaxed signatures cannot produce false negatives, but can produce false positives. We obviate false positives by assigning appropriate *weights* to terms. As a result, CUTE employs weighted term matching for classifications. CUTE automatically extracts the terms, and assigns respective weights using a learning set of network flows. We provide the theoretical foundation of our method, together with required definitions and theorems, in Section V.

The first step in CUTE extracts potential terms from sample flows. We extract terms of certain length (say of at least b bytes), which are shared between a considerable fraction of flows (say with a frequency of at least δ). Such common terms (for the formal definition refer to Definition 1) can be extracted by aligning each pair of flows, and extracting all common substrings of at least b bytes.

Considering the fact that payloads mostly start with important terms, we employ a simple tweak to distinguish these important terms; we insert a mark (*e.g.,* `|`) before the first byte of flows. For instance, when we insert `|` at the beginning of flows, we will extract `|GET` instead of `GET` which is probably unique for `HTTP`, and `HTTP`-based protocols. This simple tweak significantly improves the experimental results as illustrated in Section III.

The next step is assigning weights to terms. The main idea behind CUTE's weight function is that we should assign higher weights to terms that are unique to a protocol. In other words, terms which are frequent in different protocols are not important, nor useful, for classification. We assign weights using the weight function presented in Equation 1:

$$W_t^p = \begin{cases} \left(\frac{f_t^p}{\sum_{p \in P} f_t^p}\right)^\rho & f_t^p \geq T \\ 0 & f_t^p < T \end{cases}, \quad (1)$$

where P, p, t are, respectively, the set of all protocols, a sample protocol, and a sample term. Moreover, f_t^p and W_t^p, respectively, denote frequency and weight of term t in protocol p. We note that T and ρ are the parameters of this weight function. Section III discusses appropriate values for T and ρ. Note that Equation 1 produces weights of 1 for unique terms.

The last step is to classify network flows. CUTE searches for the weighted terms of each protocol within flow payloads. The similarity of a flow to a protocol is simply defined as the *average weight* of matched terms in the flow payload. Then, the protocol with the highest similarity is selected as label protocol of the flow. Note that we have evaluated CUTE using other alternatives (*e.g.,* using the maximum weight) and discovered that the presented version achieves the best accuracy. Those alternatives are eliminated due to lack of space.

CUTE's classification algorithm has two inputs: (*i*) the flow payload f, and (*ii*) the set of top r-percentile weighted terms for each protocol ($W = \{W_{p_1}, W_{p_2}, \cdots, W_{p_m}\}$) ($r$ is a parameter of CUTE). Using only the top terms improves accuracy and efficiency of our method, since it eliminates unimportant terms extracted due to locality, errors, and noises.

Note that we have used the naive string matching algorithm only for the sake of readability. This string matching method can be, readily, replaced with efficient techniques such as the Aho-Corasick algorithm[1].

In the next section, we illustrate the experimental results of classifying real-world network flows using CUTE.

III. Experimental Evaluation

To evaluate CUTE, we use two network datasets, each containing about 40 million flows, captured from two different tier-1 ISPs for a duration of 30 minutes at different dates. To preserve the anonymity of these ISPs, we call them ISP1 and ISP2. These ISPs are located in different continents, and have completely different traffic characteristics. To create a base of evaluation, the captured flows are classified using a set

Algorithm 1: CUTE's classification algorithm

```
Input: f ;                                   /* The Flow */
Input: W ;              /* Top r% of Weighted Terms */
Result: result_p ;      /* The detected protocol(s) */
max ← −1;
result_p ← ∅;
for W_p ∈ W do
    sum ← 0; count ← 0;
    for (t, w) ∈ W_p do
        if f contains t then
            sum ← sum + w; count ← count + 1;
    if max = sum/count then
        result_p ← result_p ∪ p;
    else if max < sum/count then
        result_p ← p;
        max = sum/count;
return result_p;
```

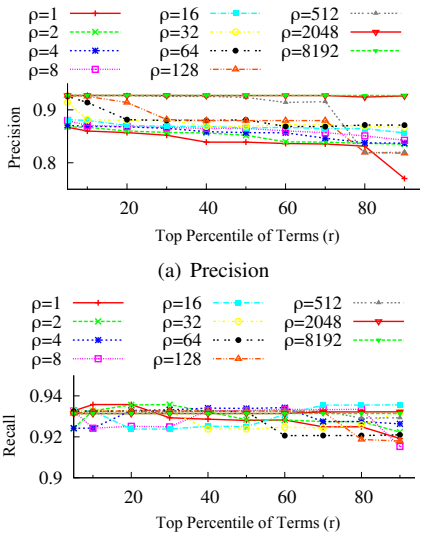
of manually-created, commercially-designed signatures, which we called baseline signatures. Note that there are flows in the dataset, which cannot be classified by the baseline signatures due to either payload encryption, flow incompleteness, or signature over-specification. We eliminate these unknown flows from the datasets.

To show CUTE's robustness, we use one data set as the learning data set, and the other one for evaluation. Note that we are able to evaluate CUTE only on protocols shared between the both datasets.

In the training phase, we extract all common substrings of at least 4 bytes as protocols' terms. Here, we can observe the first advantage of CUTE over complete matching techniques. There are many frequent, but unimportant terms, such as "date"s (*e.g.,* "Sun, 24 Feb 2010 08:49:37 GMT") in `HTTP` flows. If we extract signatures automatically, these unimportant terms would be present in protocol signatures due to their high frequency. Such signatures perform well on the learning data set, but result in poor classification, when other datasets are used. Since CUTE uses partial matching and weighted terms for classification, it simply tolerates such misleading terms.

In the experiments, we set the threshold T (Equation 1) to 0.1, as the frequencies of important terms (such as `HELO` in `SMTP`, and `POST` in `HTTP`) are about 10 percent in real traffic. Setting T to smaller values may result in more unimportant terms in classification.

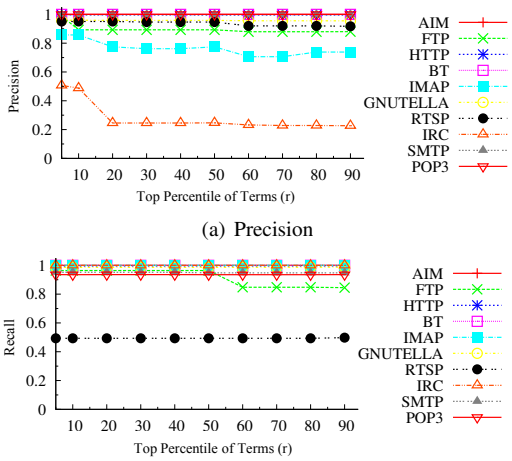
The first experiment explores the impact of ρ on CUTE's precision and recall. Figure 1(a) shows that there is a strong correlation between ρ and precision: the higher the ρ, the more unique the terms, and the more precise the classification. Figure 1(b) displays the relationship between recall and ρ. One interesting point, here, is that for lower values of ρ, the precision is more sensitive to r (the top percentile parameter). Larger ρ's result in smaller term weights except for the unique terms. For instance, terms in the top 50-percentile, when ρ is 512, are the same as terms in the top 5-percentile when ρ is 32. In fact, for any value of ρ there is a value for r that leads to the same precision while maintaining the same recall rate. Since there are different number of flows for each protocol in the dataset, Figure 1(b) does not convey a meaningful relation between ρ and recall values: for different values of ρ, recall



(a) Precision

(b) Recall

Fig. 1. Average recall and precision for different ρ values



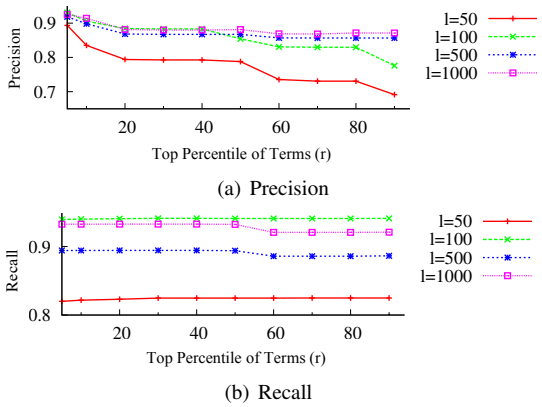
(a) Precision

(b) Recall

Fig. 2. Recall and precision for ρ = 64

values have jitters of 2%. This is mainly due to the low number of flows for some protocols in our data set.

Figure 2 shows that CUTE accurately classifies most protocols' flows. As this figure suggests, CUTE has the maximum precision when we use the top 5-percentile terms, and $\rho = 64$. In Figure 2(a), we observe that CUTE classifies all protocols with high precision, except for `IRC`. The precision for `IRC` flows is 50% using the top 5-percentile, since `IRC` flows are basically unstructured text containing terms shared with other protocols. In such protocols, even the top terms are not very important. Thus, even the top 5-percentile terms result in a considerable amount of false positives. Figure 2(b) illustrates the recall for different protocols. It shows that CUTE achieves high recalls for all protocols except `RTSP`. The recall value for `RTSP` flows is 50%, since `RTSP` and `HTTP` share a lot of terms; hence, `RTSP` does not have a sufficient number of highly weighted terms. The only terms unique to `RTSP` are its commands, *e.g.,* `DESCRIBE`.



(a) Precision

(b) Recall

Fig. 3. Average precision and recall using the first bytes of flows

In the previous experiments, we have used the first 1KB of flow payloads for classification. One of the advantages of exploiting terms instead of term lists is that a flow can be classified using its very first few bytes. To illustrate this advantage, we present classification results using the first l bytes of flow payloads. Figure 3 portrays the average precision and recall for different values of l. It shows that using more bytes from payloads generally results in higher precision and recall. Meanwhile, we can improve the efficiency of CUTE by using fewer payload bytes in classification, and still getting high precision and recall values. We observe that by using only the first 50 bytes of the flows, we achieve acceptable precision and recall. An interesting extension to this work is to find the suitable value of l for different protocols.

There is one counterintuitive situation in these graphs: the maximum average recall is achieved when $l = 100$. To investigate this problem, we take a deeper look at the detailed classification results when $l = 100$. Figure 4 displays the recall values for different protocols when $l = 100$. By comparing Figures 2(b) ($l = 1000$) and Figure 4 ($l = 100$), we observe that, when $l = 1000$, the recalls for three protocols (`HTTP`, `FTP`, and `POP3`) are higher, but it is much lower for `RTSP`. Therefore, the average recall, misleadingly, shows improvement when we reduce l to 100.

Another advantage of CUTE is its capability to classify unknown flows. To evaluate this advantage, we randomly selected 15000 unknown flows out of our data set, and fed them to CUTE. CUTE classified 60% of these flows. Since we do not have any ground truth for these unknown flows, there is no straight forward approach to calculate the accuracy. To estimate the accuracy, we have manually explored and labeled 40% of these flows. Most of these unknown flows are incomplete, partial flows. As shown in table I, most of the unknown flows are `HTTP`, and are labeled correctly by CUTE. These unknown flows are legitimate `HTTP` flows but they are either response only, request only, or from proprietary non-compliant applications.

FTP	HTTP	IMAP	BT	GT	RTSP	SMTP	IRC
0.1%	88.5%	0.2%	0.1%	1.8%	3.9%	4.6%	0.8%

TABLE I
NEW LABELS OF UNKNOWN FLOWS (5-PERCENTILE, AND $\rho = 64$)

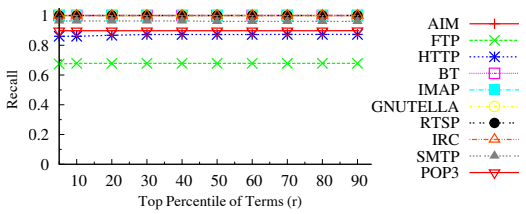


Fig. 4. Recall when $l = 100$

IV. FORMAL MODELING

In this section, we model the traffic classification problem to theoretically analyze CUTE. We aim to compare CUTE with existing automatic signature extraction methods (such as Longest Common Subsequence [21] and Laser [18]) which generate term lists (a sequences of terms) as protocol signatures. We show that CUTE is significantly faster (due to the use of partial matching instead of complete matching) without sacrificing the precision and recall rates.

In general, a term list is (*i*) a group of terms, accompanied with (*ii*) ordering constraints. Ordering constraints are to increase the accuracy of classification, but they come at the cost of matching efficiency, and complexity of automatic signature generation. In contrast, CUTE relaxes those constraints, and uses weighted terms for classification. The question is "how important are these constraints?" and "how adversely does the classification algorithm perform if we relax these constraints?". In what follows, we formalize CUTE's approach, and then theoretically analyze it.

We, first, present our definition of common terms, and then we proceed with the theoretical model for comparing our proposed signatures with existing protocol signatures, in next section. Note that we have used *term* and *common term* interchangeably in this paper.

Definition 1: Let b, and δ be given numbers. We define a continuous subsequence of bytes as a **term** iff: (*i*) it has at least b byes, (*ii*) it occurs with a probability of at least δ in flows of the respective protocol, and (*iii*) it is not a substring of another term with the same probability. $T(P)$ denotes the set of common terms for protocol P, *i.e.*,

$$T(P) = \{t_i \quad | \quad |t_i| \geq b \wedge Pr(t_i|P) \geq \delta \wedge \nexists t_j : t_i \subset t_j \wedge \mathbf{Pr}(t_j|P) = \mathbf{Pr}(t_i|P)\},$$

where $\mathbf{Pr}(t|P)$ is the probability of having term t in the flows of protocol P.

As an example, consider the `HTTP` protocol. When $b = 3$ and $\delta = 0.1$, `POST` and `GET` are among the *terms* for `HTTP`. Let us consider the character sequence `POS`. Obviously, its frequency is at least equal to the frequency of `POST`. Two scenarios may happen here: (*i*) `POS` has the same frequency as `POST`: in this case, we do not consider it as a term. (*ii*) `POS` has a higher frequency than `POST`: here, both `POS` and `POST` are considered *terms* of `HTTP`.

Terms are the basic constructs of term lists. A flow is matched with a term list, iff it contains all terms in the list with respective ordering. For instance, a flow which contains `xbxaxcx` will not match with $L = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$, but it matches

$L = \{\texttt{b}, \texttt{a}, \texttt{c}\}$. We denote this flow matching mechanism by the *complete matching operator*.

Definition 2: The complete matching operator $\oplus$, between the flow f and term list $L = \{t_1, \cdots, t_n\}$, is defined as follows:

$$f \oplus L \Leftrightarrow index(f, t_1) > 0 \ \wedge lsplit(f, t_1) \oplus \{t_2, \cdots, t_n\},$$

where $index(f, t)$ is a function that returns the first position of term t in flow f if f contains t, and -1 otherwise. Moreover, $lsplit(f, t_i)$ returns the subsequence of bytes in f from the first character after t_i to the end. For instance, $lsplit($xxabcyyabc,abc$)$ is yyabc.

Similar to terms, we may have redundant term lists. A term list L is a *sub-list* of another term list L', if L' matches all flows that could match by signature L. For instance, $\{\texttt{a}, \texttt{b}, \texttt{c}\}$ is a sub-list of $\{\texttt{b}\}$.

Definition 3: Term list L_i is a sub-list of term list L_j, $L_i \subset L_j$, iff: $\forall f, L_i \oplus f \Rightarrow L_j \oplus f$.

Similar to terms, we eliminate redundant term lists. A term list is redundant, if one of its sub-list has the same probability. The remaining term lists form the protocol signatures according to Definition 4.

Definition 4: A term list is an **ordered signature** for protocol P if:

(*i*) It matches more than T fraction of flows: $\mathbf{Pr}(L|P) \leq T$,

(*ii*) It is not redundant:

$$\forall L \nexists L', L' \neq L \wedge L' \subset L \wedge \mathbf{Pr}(L|P) = \mathbf{Pr}(L'|P),$$

where $\mathbf{Pr}(L|P)$ is the probability of term list L for flows of protocol P.

Definition 4 intuitively implies that we consider *all* important signatures for a protocol P. This is what we call the *completeness property*.

The ordered matching operator ($\oplus$) satisfies two constraints: (*i*) *full matching*: a flow must contain all terms of a term list to be matched, and (*ii*) *ordering*: the ordering of the terms should be the same in the term list and the given flow. The matching operator used in CUTE relaxes these constraints, as defined in Definition 5 In contrast to the complete matching operator, xxbxxaxxcxx matches with $L = \{\texttt{a}, \texttt{b}, \texttt{c}\}$, if we use the partial matching operator.

Definition 5: A flow is partially matched with a term list, iff it contains at least one of term from the list. The partial matching operator $\odot$, between flow f and *term list* L, is formally defined as follows:

$$f \odot L \Leftrightarrow \exists t_i \in L, index(f, t_i) > 0.$$

The partial matching operator is significantly less complex than the complete matching operator, while it might introduce inconsistencies in classification. In the next section, we discuss how poorly traffic classification algorithms perform, if we use partial matching operator $\odot$ instead of the complete matching operator $\oplus$.

V. Theoretical Analysis

In what follows, we analyze CUTE based on the model presented in previous section. In our theoretical analysis, we aim at providing bounds on CUTE's classification error, in comparison to existing automatic signature generation methods.

Existing automatic signature generation methods have an intrinsic error probability, as they extract all possible term lists with a probability of at least T for all protocols. They miss-classify flows, if some important signatures are present in less than T fraction of flows, and respective flows cannot be identified using other signatures. The total error probability of a protocol can be even unbounded, if too many tiny clusters of flows (less than $T - \epsilon$ fraction of flows) are not covered by any extracted signature. That said, these methods perform well in practice since such worst cases are too rare as long as we select an apt value for T.

The partial matching operator can introduce further errors. These errors do not include false negatives, since the partial matching operator certainly matches all flows matched by the complete matching operator. Moreover, as long as the classification does not result in *conflicts* between term lists (when one flow matches with two or more protocols), false positives are not introduced. So, the only scenario that can lead to error for partial matching is when we have conflicting signatures. Note that the correct protocol is always among the set of identified, conflicting protocols, since the partial matching operator always includes results of the complete matching operator.

We study the probability of having false positives (or equivalently conflicts), in certain scenarios, where some conditions hold. In fact, we propose two sufficient condition which bound the false positive rate introduced by a signature.

A. Definitions and Assumptions

As previously mentioned, we represent protocol signatures using term lists. The set of term lists of a protocol P is denoted by $\mathbb{L}(P) = \{L_1^P, L_2^P, \cdots, L_m^P\}$.

To analyze the error probability of the partial matching operator, we have the following simplifying assumptions:

Assumption 1: (Coverage) We can detect the protocol of every flow using existing signatures. Since we aim at studying the effects of applying the partial matching operator, it is reasonable to ignore unknown flows.

Assumption 2: (Consistency) Signatures of different protocols are conflict-free, unless one of them is a sub-signature of the other one. In such cases, the most specific signature is preferred. Note that, we allow conflicting sub-signatures to classify overlay protocols.

B. Uniqueness Condition

We start with the stronger (and simpler) sufficient condition. Suppose that each term list has a term, which is not a member of any term list in signatures of other protocols–in other words, it is unique to the respective protocol. Note that the unique terms of a protocol may exist in less than T fraction of flows of another protocol. If the Uniqueness Condition (Condition 1) holds, we can prune non-unique terms to have less conflicts with the same coverage as the original signatures.

Condition 1: The *Uniqueness* condition holds iff for every *term list* L of protocol P_i, there is at least one term t which

does not occur in any *term list* L' of other protocols; *i.e.,*

$$\forall L \in \mathbb{L}(P_i), \exists t \in L \text{ s.t. } \forall P_{j(j\neq i)} \nexists L' \in \mathbb{L}(P_j), t \in L'$$

Surprisingly, the uniqueness condition holds for many real-world protocols even though it is a seemingly strong condition. The only exceptions are overlay protocols, which share the same terms with their underlying protocol (*e.g.* BitTorrent running on HTTP). Shortly, we will introduce a weaker, more practical sufficient condition in Section V-C, based on the Uniqueness condition.

When the *Uniqueness Condition* holds, we eliminate non-unique terms to simplify signatures, and also to improve efficiency. We remove every term t with the following property:

$$\exists L \in \mathbb{L}(P_i), L' \in \mathbb{L}(P_j)_{(i\neq j)} t \in L \wedge t \in L'.$$

Having at least one unique term, each term list contains one or more terms after pruning. We simplify each term list by selecting *one* of its unique terms (later in this section, we discuss the best way to select from unique terms). Thus, our pruned term lists contain one and only one term. For each term list L, the pruned version is denoted by $\mathbb{P}(L)$. According to Theorem 1, the probability of producing conflicts between a pruned term list of a protocol, and flows of another protocols is bounded by T.

Theorem 1: If the Uniqueness Condition is satisfied, a pruned signature of protocol P_i conflicts with another protocol P_j with a probability of less than T. *i.e.,*

$$\forall P_i, P_{j\neq i}, L \in \mathbb{L}(P_i) : \mathbf{Pr}(f \odot \mathbb{P}(L) | f \in P_j) < T.$$

Proof: (Proof by contradiction) Suppose that $\mathbb{P}(L)$ is a pruned signature of protocol P_i composed of term t, and the probability of matching a flow f of protocol P_j with $\mathbb{P}(L)$ is more than T, *i.e.,* $\mathbf{Pr}(f \odot \mathbb{P}(L) | f \in P_j) \geq T$. Equivalently, the probability of having t in flows of P_j is at least T. Thus, t must be part of a signature for protocol P_j (see Definition 4) and is not unique. This contradicts with our assumption that t is unique, and the proof merely follows. ∎

Since two protocols can have several conflicting unique terms, the total probability of having conflict between two protocols can be higher than T. The probability that we mis-classify flows of a protocol P_j to P_i is $\mathbf{Pr}(f \odot \cup_{L\in P_i} L | f \in P_j)$, which is always less than $T \times k_i$ where k_i is the number of pruned signatures in P_i. Note that k_i is small in practice; hence, if the Uniqueness Condition is satisfied, we can use the partial matching operator instead of the complete matching operator with a few conflicts in practice.

By minimizing the number of terms for each protocol, we can improve the efficiency of matching: the less the terms, the faster the matching. Suppose that, we have the following term lists of unique terms for protocol P: $\{\{A, C\}, \{A, C, E, F\}, \{B, D\}, \{C\}\}$. The optimum, equivalent term lists for P, is $\{\{B\}, \{C\}\}$ since all term lists have either B or C. Thus, without rendering any further issues, we increase the performance of matching by using only two unique terms.

The question is what is the minimum number of unique terms that covers all term lists? This problem, in general, is equivalent to the Minimum Hitting-Set problem (Definition 6) which is both NP-Hard and APX-Hard [4].

Definition 6: Hitting-Set. Suppose that we have a finite set S and a collection C of subsets of S. A Hitting-Set for C is a set $S' \subseteq S$ with the following property: $\forall S'' \in C, \exists t \in S'' : t \in S'$.

The term lists, however, have an interesting property: based on Definition 4, we can easily show that the intersection of two different term lists of protocol P is either empty, or a term list of P–*i.e.,* $\forall L_1, L_2 \in \mathbb{L}(P), L_1 \cap L_2 = \varnothing \vee L_1 \cap L_2 \in \mathbb{L}(P)$. The reason is that, a non-empty intersection of two different term lists has a higher probability than T and is a signature by itself.

Based on this observation, we propose an algorithm we call the Minimum Hitting-Set for Protocols (MHSP), which is capable of discovering the minimum Hitting-Set of term lists in $O(|P|^2)$. According to Theorem 2, this algorithm returns the correct hitting set for our problem.

Algorithm 2: MHSP: Minimum Hitting-Set for Protocols

```
Data: P // the protocol
Result: Minimum Hitting-Set
Disjoint_Set ← ∅;
for L ∈ P do
    Is_Disjont ← True;
    for L' ∈ P do
        if L' ⊂ L then
            Is_Disjoint ← False;
    if Is_Disjoint then
        Disjoint_Set ← Disjoint_Set ∪{L};
Hitting_Set ← ∅;
for L ∈ Disjoin_Set do
    Hitting_Set ← Hitting_Set ∪ head(L);
return Hitting_Set;
```

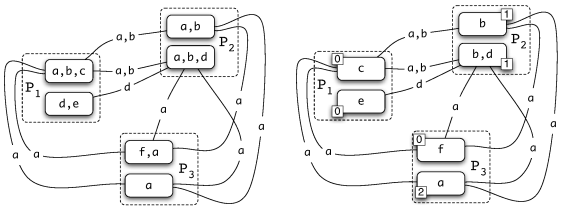
Theorem 2: MHSP finds the minimum Hitting-Set for protocol P.

Proof: Suppose that the algorithm returns a set of terms: $S = \{t_1, t_2, \cdots, t_k\}$. It is obvious that $|S| = |\text{Disjont\_Set}|$. Since there is no common terms between any two members of the Disjoint_Set, the minimum Hitting-Set cannot have less than |Disjont_Set| members. Therefore, S is one feasible minimum Hitting-Set. ∎

C. The Weak Sufficient Condition

The Uniqueness condition is too strong specially for overlay protocols. For instance, `Gnutella` shares most of its terms with `HTTP` since it is an overlay protocol; hence, `HTTP` will not have enough unique terms. To resolve this issue, we propose a looser sufficient condition which holds in almost all cases including overlay protocols, and can lessen the probability of false positives, as well.

Instead of neglecting all non-unique terms, we can assign lower weights to these terms, and classify flows based on a weighted term matching approach, *i.e.,* matched terms with the highest weights determine the protocols of a flow. Moreover, for each term least, we can keep the terms with the highest weight, as the other lower weighted terms do not affect classification.



(a) Initial Graph (b) Final Graph

Fig. 5. The Signature Graph built from $P_1 = \{\{a,b,c\},\{d,e\}\}$, $P_2 = \{\{a,b\},\{a,b,d\}\}$, and $P_3 = \{\{f,a\},\{a\}\}$

Let us illustrate this idea using an example. Suppose that we have the following protocol signatures: $P_1 = \{\{t_1,t_2,t_3\}\}, P_2 = \{\{t_2,t_3,t_4\}\}$, $P_3 = \{\{t_2,t_3\}\}$. t_1 and t_4 should have high weights since they are unique. Non unique terms, such as t_2 and t_3 on the other hand, should have lower weights. If we assign weights according to their inverse protocol frequency, and keep the terms with the highest weight for each term list, new signatures will be: $P_1 = \{\{(t_1,1)\}\}$, $P_2 = \{\{(t_4,1)\}\}$, $P_3 = \{\{(t_2,\frac{1}{3}),(t_3,\frac{1}{3})\}\}$. Using these new signatures, a flow containing t_1 is classified as protocol P_1, since t_1 is one of the two highest weighted terms. Interestingly, flows containing only t_2 or t_3 are detected as P_3. Note that weights can be assigned in different ways as long as they reflect the uniqueness of term.

To realize this idea, we propose the *Signature Graph*, which is used for (*i*) selecting the best terms from each *term list*, and (*ii*) assigning appropriate weights to them. The process of creating a Signature Graph has multiple steps. Initially, each *t*erm list is a node in this graph. An edge connects two nodes, iff they are of different protocols and share at least one term. The shared terms are the labels of the edge. Each node of the graph is assigned an integer called $mark$ which is initially 0, *i.e.,* $\forall n \in N(G), mark(n) = 0$.

Figure 5(a) illustrates the initial Signature Graph for term lists of 3 protocols: $\mathbb{L}(P_1) = \{\{a,b,c\},\{d,e\}\}$, $\mathbb{L}(P_2) = \{\{a,b\},\{a,b,d\}\}$, and $\mathbb{L}(P_3) = \{\{f,a\},\{a\}\}$. Note that all the nodes are, initially, marked as 0.

The next step is to find conflicting nodes. We mark each node with no unique term as 1. Note that a term is unique to a node, iff it is not on any edges of the respective node. We remove all non-unique terms from nodes marked as 0. Those nodes will not become empty, since they have at least one unique term. Then, in the subgraph of nodes with mark 1, we increment the mark of the nodes with no unique term to 2, and remove non-unique terms (non-unique in the subgraph) from nodes with mark 1.

We repeat this process until one of the following conditions hold: (*i*) all nodes, marked as i, are now marked as $i+1$, or (*ii*) there is no node marked as $i+1$. If the first condition holds, the process never terminates, and we are unable to build the Signature Graph. If the second condition holds, we have the final Signature Graph, with no more node to process. An example of the final Signature Graph is shown in Figure 5(b). If we can build the final Signature Graph for a set of signatures, the *Weak* sufficient condition (Condition 2) is satisfied.

Condition 2: Weak Sufficient condition holds in a set of signatures, if we can create a non-conflicting *signature graph* for them by following the pruning procedure described above.

Terms on the nodes with lower marks are more important than terms on nodes with higher marks, since marks clearly represent terms' frequencies among different protocols, or simply the inverse of terms' uniqueness. Consequently, we assign weights to the term lists, or basically nodes in the final Signature Graph, in a way that it preserves the following property: the lower the mark, the higher the term's weight – *i.e.,* $\forall t_i \in n, t_j \in n', mark(n) < mark(n') \Rightarrow weight(t_i) > weight(t_j)$. An example of such a weighting function is $(1 + mark(n))^{-1}$ for the terms on node n.

Lemma 1: If the weak sufficient condition holds, there is a term t for any node n_i, which does not exist in any other node with higher or equal mark – *i.e.,* $\exists t \in n_i; \forall n_j (i \neq j), mark(n_j) \geq mark(n_i) : t \notin n_j$

Proof: Suppose that we have a node n_i, and its mark is m. According to the pruning procedure, all terms in n_i shared with other nodes with mark m or higher are pruned. n_i cannot be empty, otherwise its mark should have been incremented to $m+1$ or a higher value. Thus, n_i contains at least one term which is not shared with nodes of equal or higher marks. ■

Lemma 2: If $L_1 \subset L_2$, then $mark(L_2) < mark(L_1)$.

Proof: Simply follows from Lemma 1. ■

As previously mentioned, if a flow matches with several term lists, only the term lists of the highest value are considered as correct matches. In other words, if a node with mark m matches a flow, no nodes with a mark higher than m are considered as a match.

As stated in Theorem 3, if we use the terms on the nodes of a final Signature Graph, we can partially match traffic with a false positive rate of less than T.

Theorem 3: Using the weighted terms from a successfully processed Signature Graph G, the probability of matching a term t with flows of protocol P_j is less than T.

Proof: Suppose that we have a false positive, and a flow f is matched with $t \in P_i$, but its should be matched with $t' \in P_j$. Suppose that nodes n_i, and n_j are representing t and t' in the signature graph, respectively. There are two possible cases: (*i*) $mark(n_j) < mark(n_i)$, and (*ii*) $mark(n_j) \geq mark(n_i)$.

According to Lemma 1, if n_i's mark is larger than n_j's mark, n_j has a term t, which is not shared with n_i. On the other hand, we know that if a flow matches both n_i and n_j, we will choose n_j as the matching signature, since its terms have larger weights. Therefore, in Case (*i*), f does not have any of the unique terms in n_j. Hence, it does not match respective term list, and it contradicts with our assumption.

If $mark(n_j)$ is greater than or equal to $mark(n_i)$ (Case (*ii*)), we know that there is a term on n_i, which is not pruned and is not a member of n_j. We know that t occurs in less than T fraction of P_j's flows, otherwise it was part of the protocol signature. Thus, the probability that a given flow f matches with t is less than T, and the proof follows. ■

Lastly, according to Theorem 4, the Weak Sufficient Condition is a general case of the Uniqueness Condition.

Theorem 4: Uniqueness condition is a special case of *weak sufficient condition*. In other words, if *uniqueness condition* holds, *weak sufficient condition* also holds.

Proof: If the *uniqueness condition* holds, there should exist at least one unique term in each signature, and no two signatures in the list can be conflicting. Therefore, the signature graph of these sets of signatures shows no conflict between nodes and the *weak sufficient condition* holds. ■

The properties of the Signature Graph shows that, if we have a set of appropriately weighted terms for a set of protocols, and the terms' weights reflect their uniqueness, we can have an accurate and efficient traffic classification method. This is in agreement with our our experimental evaluation of CUTE, and shows the intuition behind Signature Graph's role in CUTE.

VI. Related Works

Deep packet inspection methods [15], [14], [10], [21] are generally complex, and time-consuming, but very accurate. One of the main problems of these approaches, besides their complexity, is that they highly rely on manually designed signatures. To solve this problem, quite a few efforts has been made to generate protocols signatures automatically. Longest Common Subsequence (LCS) [21] is a well-known approach to generate signatures. The LCS method searches for the longest subsequence common in all or the majority of the flows of a particular protocol and uses that subsequence as the protocol's signature.

Another problem with DPI methods is that there is always a trade-off between memory usage and the speed of regular expression matching. Efforts in this area has focused on rewriting signatures and grouping schemes (*e.g.,* [23]), narrowing regular expression definitions (*e.g.,* [6]), or utilizing hardware-based methods (*e.g.,* [24]).

Feature-based classification methods, on the other hand, take advantage of layer 4 information and machine learning techniques to classify network traffic [2], [16], [8], [9], [5], [7], [20], [3], [13]. These methods are all automatic, but they are prone to be over-trained [19]. We note that DPI methods are considered as more accurate approaches, and generally form the ground-truth for feature-based methods. An automatic efficient DPI method is a better alternative, whenever the flow payload is not encrypted.

Worm Detection is another area of research which is correlated with Traffic Classification. *Payload-based* worm detection techniques, which try to match packets' payload against a set of signatures to detect potential attacks, are very similar to DPI methods. Several attempts have been made in this area to automatically generate signatures for detecting worms. Among them, we can mention Honeycomb [12], Autograph [11], Earlybird [22] for non-polymorphic worms and Polygraph [17] for polymorphic worms. Note that a polymorphic worm is a type of worm which can use self-encryption mechanisms, and semantics-preserving code manipulation techniques to evolve as spreading in the network.

In Polygraph, J. Newsome *et al.* have shown that all worms have some invariant parts although they can be small. Using this property, they have proposed three methods (*Conjunction*, *Token-Subsequence*, and *Bayesian*) to detect attacks of polymorphic worms. While *conjunction* signatures try to match flows with several tokens without considering the order of these tokens, token-subsequence signatures consist of a list of ordered tokens.

In Token-Subsequence approach, which is extended in Laser [18] for traffic classification, a list of indivisible tokens (terms in our terminology) is extracted from the flows, and each flow is represented with a list of its tokens. For each protocol, the Token-Subsequence method tries to find one maximal substring which exists in all sample flows. In its multiple-signature version, sample flows are clustered and one signature is generated for each cluster. The signature for each cluster should match all flows inside it. In Ideal case, each cluster contains the flows of one protocol and the signatures are non-conflicting. Bayesian signatures, on the other hand, assign a weight to each token and a flow will be labeled as a worm if sum of the matched tokens' weights is larger than a predetermined threshold. The experimental results show that Bayesian signatures are equivalent to one special substring, *"best substring"*, as far as accuracy of classification is concerned [17]. However, no reason has been provided for this phenomena. Moreover, they have not suggested any approach to extract the best substring. The best substring is only reported in Polygraph after brute-forcing all possible substrings.

There are some similarities between token-subsequence and *term lists* in this paper. In both approaches, in an ideal case, a set of non-conflicting term lists (see Assumption 2) with a specific minimum probability (T in CUTE and S "the size of the smallest cluster" in Token-Subsequence) is created. Since these signatures are non-conflicting, false positive in both approaches is zero. Note that false positive may increase when we use tokens instead of token subsequences in Polygraph, and terms instead of term lists in CUTE.

Despite those similarities, CUTE and Polygraph have three important differences. First, CUTE, automatically, extracts the best substrings, and we theoretically explain why it has similar accuracy to Bayesian signatures of Polygraph. Second, CUTE produces less false negatives compared to token-subsequences in Polygraph, since CUTE extract all possible terms while token- subsequence is limited by the generated clusters. Third, CUTE uses partial matching which is more efficient and has a higher recall rate, but Polygraph uses complete matching, or complete unordered matching for conjunction signatures.

VII. Discussion

Exploiting terms for classifying network flows has some advantages and some drawbacks. The major drawback is generating false positives. Based on theoretical analysis and experimental evaluations, our method is capable of classifying most of todays' protocols with a negligible amount of false positives. Accordingly, these simple signatures are competitive alternatives to traditional complex signatures: they are simpler, more efficient, and easier to maintain. Moreover, they result in considerably higher recall rate.

Protocol signatures are precisely defined to classify network traffic with a high accuracy. These signatures require quite a few packets to classify a flow. Although it may increase the classification accuracy to wait for packets on the fly, it has two disadvantages: (*i*) it is more useful to discover flow protocols as soon as possible, not in middle or final stages, and (*ii*) it

might need a considerable amount of memory to keep the state of each flow for each protocol signature for a large fraction of flow's life-time. The experimental results show that CUTE can classify flows using the first bytes (50 to 100 bytes) with more than 80% of precision and recall. Moreover, CUTE can improve the classification accuracy as new packets arrive.

CUTE's output can also be enhanced by passing the conflicting results to an existing DPI methods. Accordingly, CUTE, accurately and efficiently, classifies most of the flows, and rare conflicts are resolved using a regular expression.

Signature extraction methods generating *term lists* are prone to be over-trained and generate wrong signatures. There are several misleading terms due to capturing methodologies. For instance, if all captured `HTTP` flows in the training data set are initiated in 2010, automatic signature generation methods, such as polygraph, incorporate such trifling, misleading terms into the generated signatures. Similarly, CUTE report both misleading terms (if they are sufficiently unique) and important terms (*e.g.,* `HTTP/1.1`) as `HTTP`'s terms. However, using partial matching, CUTE is able to correctly classify a legitimate `HTTP` flow without those trifling terms (*e.g.,* 2010).

Another important aspect of all automatic signature generation methods, including CUTE, is to analyze their robustness to adversarial attacks. One possible attack to term-based methods is to create new fake protocols which contain other protocols' terms. This attack decreases the weight of important terms of various protocols; hence, the classification method cannot distinguish between existing real protocols and new fake ones. We note that this kind of attacks is, intrinsically, applicable to any payload-based classification method. Having said that, the partial matching is less prone to such errors compared to ordered and complete signatures, except if the adversary generates attacks which are mainly designed for partial matching (*e.g.,* by creating flows which contains the same terms with different orders). Finally, we note that CUTE is only affected by such attacks if the training data set contains those adversarial flows.

VIII. Conclusion

In this paper, we propose CUTE, an efficient DPI approach based on a set of weighted terms. Weighted terms are much simpler than regular expressions in terms of matching. Interestingly, this simplification does not come at the cost of accuracy. Our theoretical analysis illustrates that, under practical assumptions, the error probability of CUTE is negligible in practice compared to complex regular expressions.

We have also evaluated CUTE using packet traces from two tier-one ISPs. To ensure the validity of our experiments, we have used one ISP for extracting terms, and one for evaluating CUTE. Our experimental evaluation suggests that appropriately weighted terms result in highly accurate classification of network traffic. The results are close to classification results generated using complex, manually created protocol signatures. Moreover, CUTE is able to classify a considerable amount of flows which are considered as unknown by the manually created signatures.

References

[1] A. Aho and M. Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18(6):340, 1975.

[2] M. Arlitt, J. Erman, and C. Williamson. Offline/realtime traffic classification using semi-supervised learning. *Performance Evaluation*, 64(9-12):1194–1213, 2007.

[3] T. Auld, A. Moore, and S. Gull. Bayesian neural networks for internet traffic classification. *IEEE Transactions on Neural Networks*, 18(1):223–239, 2007.

[4] G. Ausiello, A. D'Atri, and M. Protasi. Structure preserving reductions among convex optimization problems. *Journal of Computer Systems and Science*, 21(1):136–153, 1980.

[5] L. Bernaille, R. Teixeira, and K. Salamatian. Early application identification. In *Proce. of the ACM CoNEXT conference*, pages 1–12, 2006.

[6] P. Bille and M. Thorup. Regular expression matching with multi-strings and intervals. In *Proc. of the ACM-SIAM Symposium on Discrete Algorithms*, pages 1297–1308, 2010.

[7] M. Crotti, M. Dusi, F. Gringoli, and L. Salgarelli. Traffic classification through simple statistical fingerprinting. *ACM SIGCOMM Computer Communication Review*, 37(1):16, 2007.

[8] J. Erman, M. Arlitt, and A. Mahanti. Traffic classification using clustering algorithms. In *Proc. of the 2006 SIGCOMM workshop on Mining network data*, page 286, 2006.

[9] J. Fan, D. Wu, A. Nucci, R. Keralapura, and L. Gao. Identifying hidden voice and video streams. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 7344, page 14, 2009.

[10] P. Haffner, S. Sen, O. Spatscheck, and D. Wang. ACAS: automated construction of application signatures. In *Proc. of the 2005 ACM SIGCOMM workshop on Mining network data*, page 202, 2005.

[11] H. A. Kim and B. Karp. Autograph: Toward automated, distributed worm signature detection. In *Proc. of the 13th conference on USENIX Security Symposium*, page 19, 2004.

[12] C. Kreibich and J. Crowcroft. Honeycomb: creating intrusion detection signatures using honeypots. *ACM SIGCOMM Computer Communication Review*, 34(1):51–56, 2004.

[13] Z. Li, R. Yuan, and X. Guan. Accurate classification of the internet traffic based on the SVM method. In *IEEE International Conference on Communications*, pages 1373–1378, 2007.

[14] J. Ma, K. Levchenko, C. Kreibich, S. Savage, and G. Voelker. Unexpected means of protocol inference. In *Proc. of the 6th ACM SIGCOMM conference on Internet measurement*, pages 313–326, 2006.

[15] A. Moore and K. Papagiannaki. Toward the accurate identification of network applications. *Passive and Active Network Measurement*, pages 41–54, 2005.

[16] A. W. Moore and D. Zuev. Internet traffic classification using bayesian analysis techniques. *ACM SIGMETRICS Performance Evaluation Review*, 33(1):50–60, 2005.

[17] J. Newsome, B. Karp, and D. Song. Polygraph: Automatically generating signatures for polymorphic worms. In *IEEE Symposium on Security and Privacy*, page 226, 2005.

[18] B. C. Park, Y. J. Won, M. S. Kim, and J. Hong. Towards automated application signature generation for traffic identification. In *Proc. IEEE/IFIP Network Operations & Management Symposium*, pages 160–167, Salvador, Brazil, 2008.

[19] M. Pietrzyk, J.-L. Costeux, G. Urvoy-Keller, and T. En-Najjary. Challenging statistical classification for operational usage: the adsl case. In *Proc. of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 122–135, 2009.

[20] M. Roughan, S. Sen, O. Spatscheck, and N. Duffield. Class-of-service mapping for QoS: a statistical signature-based approach to IP traffic classification. In *Proc. of the 4th ACM SIGCOMM conference on Internet measurement*, pages 135–148, 2004.

[21] S. Sen, O. Spatscheck, and D. Wang. Accurate, scalable in-network identification of p2p traffic using application signatures. In *Proc. of the 13th international conference on World Wide Web*, page 521, 2004.

[22] S. Singh, C. Estan, G. Varghese, and S. Savage. Automated worm fingerprinting. In *Proc. 6th conference on Symposium on Opearting Systems Design & Implementation*, 2004.

[23] F. Yu, Z. Chen, Y. Diao, T. V. Lakshman, and R. H. Katz. Fast and memory-efficient regular expression matching for deep packet inspection. In *Proc. of the ACM/IEEE symposium on Architecture for networking and communications systems*, pages 93–102, 2006.

[24] F. Yu, R. H. Katz, and T. V. Lakshman. Gigabit rate packet pattern-matching using tcam. In *Proc. of the 12th IEEE International Conference on Network Protocols*, pages 174–183, 2004.