Reconciling Multiple Matches for the Signature-Based Application Identification

Justin Tharp[1], Jinoh Kim[1], Sang C. Suh[1], and Hyeonkoo Cho[2]
[1]University of Texas A&M – Commerce, Department of Computer Science, Commerce, Texas 75428, USA
[2] Sysmate Inc., 1290 Dunsan-Dong Seo-Gu, Deajeon, 302-830, Korea
Email: jtharp2@leomail.tamuc.edu, jinoh.kim@tamuc.edu, sang.suh@tamuc.edu, hkcho@sysmate.com

***Abstract*—** Accurate application identification is one of the core elements of network operations and management to provide enhanced network services and security. While the *signature*-based approach that examines packet content for identification is attractive with greater accuracy than the traditional technique relying on TCP *port* numbers, one potential challenge is *multiple matches* arising when more than a single application identifies the data stream in question. In that case, the input stream cannot be adequately classified solely by the help of the application signatures, and it is necessary to establish an additional process that *reconciles* such multiple matches in order to make the final identification decision. In this paper, we address the problem of multiple matches by developing a set of *selection heuristics* that help accurately identify the application associated with the input data stream. The heuristics choose one out of a set of applications using their own unique discrimination function, and the input traffic can be classified into the selected application. Our experimental results with a recent traffic data set show that our proposed method successfully deals with multiple matches, achieving a high degree of identification accuracy up to 99% with respect to precision and recall.

***Index Terms*—**Application identification, application signatures, multiple matches, network operations and management

I. Introduction

Accurate application identification is one of the core elements of network operations and management for various purposes, including network planning and capacity provisioning, QoS assurance, security policy management and enforcement, network usage and performance monitoring, traffic engineering, and so forth [1]-[4]. A classical technique for classifying network traffic is to use TCP/UDP *port* numbers, but it has become inaccurate and unreliable due to the fact that applications can be allocated to ports dynamically and some applications can be assigned to other port numbers outside their own. For instance, BitTorrent allows users to choose port numbers other than the default port numbers between 6881 and 6889 [5]. In addition to the use of non-standard port numbers, tunneling such as HTTP tunnels makes it complicated to identify corresponding applications from the traffic only based on port numbers [3]. For these reasons, classifying traffic has become more difficult and new methods have been in development.

An alternative method widely considered is to examine *packet content* and search *common patterns* also known as "signatures" that repeatedly appear in packets belonging to a specific application. For instance, a string "0x13BitTorrent protocol" is a common pattern for BitTorrent [6].A set of collected signatures could be used for identifying applications associated with input traffic in the future. Creating signatures is often performed with a training data set ("training"), and classifying input traffic takes place with the signature set ("identification"). This signature-based approach is known to be highly accurate compared to other techniques relying on packet header information and/or statistical information of input streams such as the number of bytes and time interval for transport layer sessions (e.g., TCP connections), which makes it attractive despite the high degree of computational complexity for deep packet inspection.

One potential challenge when employing the signature-based technique for application identification is multiple *matches* arising when more than a single application identifies the data stream in question. For instance, an input stream may have a string that matches with a signature for application X, and at the same time, it may also include a signature for application Y anywhere else in that data stream. When this happens, it is hard to tell what application the input stream belongs to, since it contains signatures for both X and Y. From our experiments with existing signature-based classification techniques including LCS (Longest Common Subsequence) [7], LASER [8], and AutoSig [9], we observed a substantial number of multiple matches during the identification process, as will be presented in Section III in detail. In that case, the input data stream cannot be adequately classified solely by the help of the application signatures, and it is necessary to establish an additional process that reconciles multiple matches in order to make the final identification decision.

In this paper, we address the problem of multiple matches to help accurate identification of applications. To the best of our knowledge, there is no prior research that investigated this problem. To this end, we establish a set of *heuristics* that help battle multiple matches and

Manuscript received August 10, 2013; revised October 19, 2013.
This work was supported by the IT R&D program ofMOTIE/KEIT under Grant No. 10041548.
Corresponding author email: jinoh.kim@tamuc.edu.
doi:10.12720/jcm.8.12.883-892

identify the input stream accurately. The heuristics choose one out of a set of applications using their own unique discrimination function, and the input traffic can be classified into the selected application. Our experimental results with a recent traffic data set show that our proposed method successfully deals with multiple matches, achieving a high degree of identification accuracy up to 99% with respect to *precision* and *recall*, commonly used for measuring identification performance.

The main contributions in this paper can be summarized as follows:

- We formulate a problem of multiple matches in the application identification stage, with our observations that a substantial number of identification attempts (e.g., based on string matching) resulted in multiple matches.
- We develop selection heuristics that help accurately identify the application associated with the input stream in question. The heuristics choose one out of a set of applications using their own unique discrimination function to reconcile multiple matches.
- We present our experimental methodologies and results with a recently collected traffic data set. To evaluate the proposed heuristics thoroughly, we conduct an extensive set of sensitivity study with a broad range of parameters employed by the heuristic functions.

This paper is organized as follows. In the next section, the summary of previous research efforts closely related to this work will be provided. In Section III, we provide a formal description of the problem of multiple matches for the signature-based identification, and selection heuristics that handle multiple matches will be presented in Section IV. Section V and VI will demonstrate our experimental methodologies and evaluation results. We will conclude our presentation in Section VII.

II. Related Work

We introduce some of previous research efforts that are closely related to our work presented in this paper.

LCS [7] is an algorithm that compares two strings and finds the longest common subsequence in the two strings. For example, for two strings of "AABBCCDD" and "ABCDFFGG", LCS would return "ABCD" as the longest common subsequence because those characters appear in the same order in both strings. The resulted subsequence characters do not have to be in the same place in both strings, but they just have to appear commonly in both strings. LCS is used for many applications in a variety of domains such as gene sequence alignments [10] natural language processing [11], and network security [12]. We consider LCS for network traffic classification in this study.

LASER [8]is a modified LCS algorithm that searches for the longest common substring as signatures for traffic classification. Instead of finding the longest common subsequence, LASER searches for the longest chain of characters together that form a substring that can be found in two network streams, and produces an "exact" string in which the characters should be in the same place in both strings, whereas an LCS signature is not necessarily exact and more like a pattern.

AutoSig [9] also searches for common substrings in flows but does not rely on the LCS algorithm. It instead breaks a flow into multiple chunks (called "shingles") that are compared and merged together to generate signatures. It also uses a tree to separate the substrings into further substrings so that all possible signature combinations can be recognized and used for flow identification. Like LASER, AutoSig produces a set of "exact" strings as application signatures.

The authors in [13]incorporated several techniques for signature creation for application identification. The input traffic data go through several processes in which first the data is extracted and tokenized, followed by a sequence alignment technique, and finally the signatures are constructed once all the data have been processed. Similar with LCS, it does not assume exact strings for application signatures but uses regular expression to represent signatures.

ACAS [14] also utilizes packet content (i.e., application information) for traffic classification, but it is different from the above techniques that search for common subsequences or common substrings that are found in network traffic. ACAS rather relies on machine learning algorithms for classifying network traffic. To enable this, the first N bytes are encoded and used for training, based on which the application identification takes place with machine learning techniques such as AdaBoosting. The authors showed promising results with a high degree of accuracy, but our work more considers traditional string matching for network traffic classification.

III. Problem Description

In this section, we formulate the problem of multiple matches that we focus on throughout this paper. We also present our observations that a substantial number of identification attempts resulted in multiple matches, which motivated us to pursue this research.

A. Problem Statement

Suppose a set of applications, $A = \{A_i \mid i>0\}$, to which we classify input data streams. In this work, we consider a *unidirectional flow* as the basic unit of classification. By convention, a flow consists of a set of packets that have identical five tuples — source and destination IP addresses, source and destination port numbers, and protocol type. From a given set of flows for an application, $F = \{ f_i \mid i>0\}$, we collect a set of signatures, $S = \{s_i \mid i>0\}$for *that* application. Since we assume multiple applications for classification, we use F_X and S_X for a set of flows and signatures respectively for

application X. Note that we use the cardinality notation to represent the number of elements in a set, e.g., $|S_X|$ for the size of the signature set for application X.

The identification problem is to map the input flow f in question to an application A_i, and the mapping function ψ is defined formally as follows:

$$\psi: f \rightarrow A_i$$

The function ψ utilizes the known signature set for all applications for mapping, that is, $S_{ALL} = \{S_A, S_B, \ldots\}$.

For a given input flow f, we denote by $s_i \vdash f$ if s_i is found in f (i.e., the signature matches with the flow). Let $M(f)$ be a set of signatures, the element of which holds $m_i \vdash f$, for all i. If the size of $M(f)$ is one (i.e., $|M(f)|=1$), we refer to it as a "single" match, while it is considered as a "multiple" match if $|M(f)|>1$ holds. The otherwise case is "non-match" and the input flow will be classified into "unknown", since no signature is found from the input flow.

For a single match (i.e., $|M(f)|=1$), the identification procedure is straightforward as the mapping function ψ could simply map f to the application to which the signature in $M(f)$ belongs. For a multiple match (i.e., $|M(f)|>1$), however, we should consider *N:1* mapping since there could be more than a single application in question. We define a candidate set C as a set of applications to which any element of $M(f)$ belongs. As an example, suppose a case of five matches for f, i.e., $M(f) = \{m_1, m_2, m_3, m_4, m_5\}$. If we suppose m_1 is an element of S_X, m_2 is an element of S_Y, and the rest of the elements in $M(f)$ belongs to S_Z, then the candidate application set C should consist of $C = \{X, Y, Z\}$. This means that the mapping function ψ should have the capability to discriminate one out of the others in the candidate set for identification in the event of multiple matches, and it can be reduced to a *selection* problem.

To deal with this, we establish a heuristic function H that picks one element from the given candidate set C, as follows:

$$H: C \rightarrow c, \text{ where } c \in C$$

Suppose the real application for f is X. If $c = X$, then we say that the identification is correct; otherwise, incorrect.

B. Motivation

We now present why we are interested in this problem with our observations. We conducted a set of experiments with ten applications from a recent traffic data set. The data set we used will be described in Section V in detail. We then selected 100 flows randomly without any preference for each application from the data pool. We considered three techniques, LCS, LASER, and AutoSig, which have often been referred by past work [13], [15], [16] in the experiments. Note that our focus is not on comparing the quality of signatures for the existing techniques, but on exploring the problem of multiple matches. For this reason, we simply used the identical flow set for training and identification in our experiments.

To construct a signature set, we examined the first N bytes from each flow. We employed the default values defined by each technique. For the LASER algorithm, the number of packet constraint was set to 100 and the minimal length constraint used was 4 because the authors did not present optimal values for those parameters. For LCS, we used two additional thresholds to consider the minimal signature length and the coverage: Any candidate signature that has the byte length greater than or equal to the minimal signature length threshold was only accepted, and the coverage threshold defines the minimal fraction that the candidate signature should appear for that application. If the pattern is observed more widely than the coverage threshold, the candidate signature is accepted. A candidate signature that meets both threshold constraints is finally selected as a signature for the application.

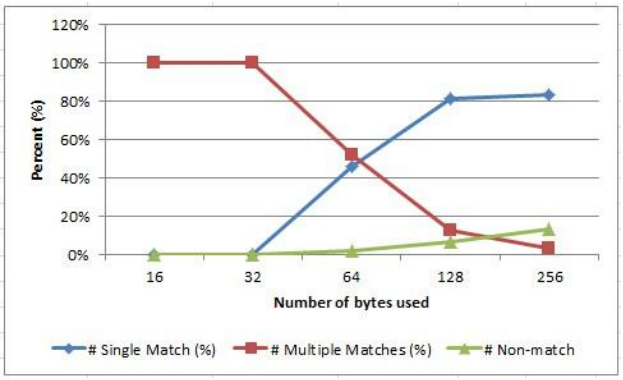

Fig. 1. Fraction of single vs. multiple matches (%)

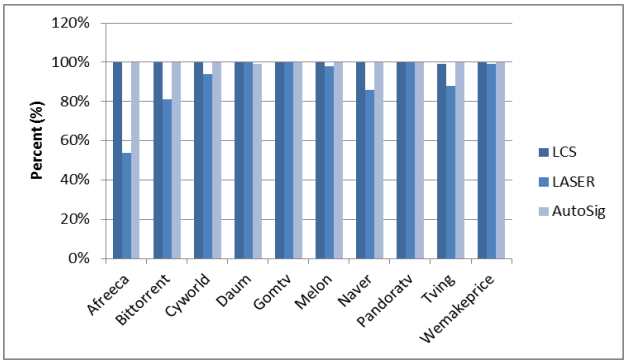

Fig. 2. % That the candidate set C includes the "real" application

The constructed signature set is then used as an input to the identification process. For identification, we considered "exact matching" for LASER and AutoSig but not for LCS, since LCS is more based on pattern matching. The exact matching means that if the signature appears in the input flow with the exact byte order without skipping, we considered the signature matches with the flow in question. In contrast, we employed the LCS algorithm again for identification when using LCS. Therefore, if the result of the LCS algorithm with the signature and the input flow is exactly the same as the signature itself, we considered that the flow includes the signature (i.e., strcmp (signature, LCS (flow, signature)) == 0?"matched":"not matched").

Fig. 1illustrates the fraction of single and multiple matches over the number of bytes examined for both signature construction and identification. When we inspect a relatively small number of bytes (i.e., N=16 and N=32), we can see a dominant number of multiple matches from the figure. As N increases, the number of single matches also increases. For N=64 bytes, almost 50% of the total identification attempts incurred multiple matches. The fraction of single matches further increases with a greater N, as shown in the figure. At the same time, the number of non-matches also increases significantly as N goes up, which is critical to identification performance because we are unable to classify the traffic if there exists no signature that matches with the input flow. Therefore, it is very important to manage N reasonably. Since it has become more important to identify applications as early as possible for real-time analysis and response [1], [14], [4] and it would be much beneficial for avoiding the number of non-matches, we consider a relatively small number of bytes for inspection in this work. Consequently, effective handling of multiple matches should be essential in the identification process.

So the key question we now have is *how can we settle multiple matches to reach the final identification?* The good news is that the quality of signatures is fairly acceptable when N=64. Fig. 2 shows the fraction whether the candidate application set (i.e., C) contains the "real" application to which the flow truly belongs. As can be seen from the figure, almost 100% of the total candidate sets contained the real applications when we use either LCS or AutoSig. This means that if we could successfully address the problem of multiple matches, we can expect a high degree of identification accuracy even with the traditional signature-based identification techniques.

These observations motivated us, and in this paper we address the problem of multiple matches. In the next section, we will present how we identify the application associated with a traffic flow even in the event of multiple matches.

IV. Reconciling Multiple Matches

To effectively deal with multiple matches, we designed the following three heuristics: greatest number of matches (GREATEST), fraction-based selection (FRAC), and probabilistic method (PROB).In this section, we describe how the heuristics could make a selection from a given set of candidate applications. The presented heuristics will then be evaluated in the followed sections.

A. Greatest Number of Matches(Greatest)

This technique chooses the candidate application that has the greatest number of application signatures that identified the flow. Formally, we assume a counter set associated with the candidate set. That is, for the candidate set $C = \{c_1, c_2, \ldots, c_k\}$, there is an associated counter set $N = \{n_1, n_2, \ldots, n_k\}$, where n_i is the number of signatures for application c_i that matched with the input flow. Based on the counter set information, the heuristic selects an application that has the greatest number of application signatures that match with the flow. The heuristic is defined as:

$$H_{GREATEST}: \mathrm{C} \rightarrow c_i, such\ that \arg\max_{1 \le i \le k} n_i$$

For example, say 5 signatures from application X identified the flow and 3 signatures from application Y identified the flow. GREATEST would like to identify the flow as application X, since more of its signatures identified the flow than the signatures from Y did.

B. Fraction-based Selection (Frac)

The above technique (GREATEST) is fairly intuitive, since it performs the discrimination based on the number of signatures that match with the given flow. However, there could be a degree of discrepancies with respect to the number of signatures each application has. In the above example, application X may have a greater number of signatures than application Y. Then there might be a higher possibility that X's signatures can be found from the input flow than Y's.

To consider the discrepancies in terms of the number of signatures for each application, we design another heuristic, fraction-based selection (FRAC), which takes a fraction into account to discriminate the candidate applications. The fraction is determined by the number of application signatures that identified the flow divided by the total number of signatures for that application. That is, we consider a set of signatures for each application in addition to the counter set N defined in the previous heuristic. As in Section II, we use S_X as the signature set for application X, and the cardinality of that is the number of elements in the set ($|S_X|$). The heuristic is then defined:

$$H_{FRAC}: \mathrm{C} \rightarrow c_i, such\ that \arg\max_{1 \le i \le k} \frac{n_i}{|S_{c_i}|}$$

For example, if 5 out of the 15 signatures from application X identified the flow, the fraction for X becomes 0.2. Similarly, if 3 out of 6 signatures from application Y identified the flow, the fraction Y will be 0.5. In that case, FRAC will identify the flow as application Y, as Y has the greater fraction (i.e., 0.5 > 0.2), while GREATEST would select X than Y.

C. Probabilistic Selection(Prob)

The last heuristic we developed relies on a simple probability model to make selection. Suppose a set of signatures $M_X = \{m_1, m_2, \ldots, m_k\}$ that match with the input flow for application X. We define p_i as the probability that the signature m_i appeared in that application, computed with the training data set; that is, $p_i = m_i / |S_X|$. We then define the collective probability P_X as the probability that the input flow belongs to application X.

$$P_X = 1 - \prod_{1 \le i \le k} (1 - p_i)$$

Thus, P_X stands for the probability that any of the matched signatures for application X appearing in the flow belongs to X. We also refer the collective probability *P* to *confidence*. Then we may want to choose a candidate with the maximum confidence (*P*), and based on the intuition, we define a heuristic as:

$$H_{PROB}: \mathrm{C} \rightarrow c_i, such\ that \arg\max_{1 \le i \le k} P_i$$

The heuristics developed in this section choose one application out of the given candidate application set to reach the final identification with their own unique discrimination function. In any event of tie that the discrimination function results in more than a single application, the heuristic simply runs a random function to break the tie.

V. Experimental Methodologies

We thus far discussed the problem of multiple matches and the selection heuristics that can settle multiple matches. We next evaluate the proposed heuristics. We introduce our experimental methodologies in this section, and then present experimental results in the next section.

A. Description of the Data Set

The traffic data set was collected in the period of March2011 – December 2011 in a LAN located in Korea, exclusively configured for data collection. Fig. 1 shows the configuration of the LAN. As seen from the figure, the LAN is connected to the Internet via a dedicated router, and client machines including PCs, laptops, and smart phones are configured via wired or wireless Ethernet. The client machines generate service requests for targeted applications to collect, and the data collector captures user packets in promiscuous mode capturing traffic for both directions. Fig. 3 shows the logical illustration of capturing packets at the data collector from both wired and wireless networks.

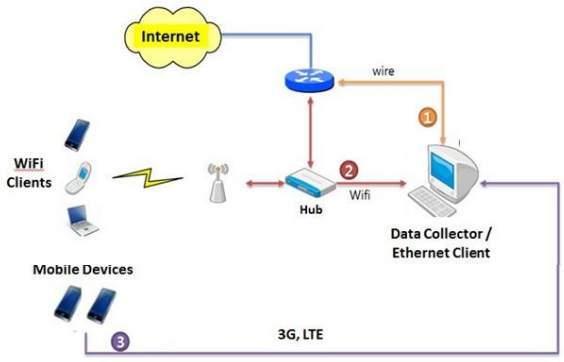

Fig. 3. Data collection network configuration: The data collector gathers captured traffic via (i) the wired Ethernet, (2) the WiFi network, and (3) 3G and LTE.

In addition to collecting data from the LAN, mobile devices were also used for data collection. The mobile devices launch applications (by using apps) that access the Internet via 3G or LTE, and captured traffic for the applications by using a packet sniffing app [17]. The collected data in the mobile devices were manually copied to the collector machine. The collected data files have the p cap format [18]. The p cap files were inspected, and any noise (including non-application traffic such as background maintenance ICMP messages) had been eliminated. The total data size is 12.5GB with 930 p cap files for 98 applications.

TABLE I. Applications used in the experiments

Application	Category	# Total Flows
BitTorrent	P2P	3,676
Cyworld	Social computing	628
Daum		1,915
Naver		1,381
Wemakeprice		523
Afreeca	Streaming	1,029
GomTV		940
PandoraTV		731
Tving		537
Melon		685

From the collected data, we obtained flow information based on the five tuples (i.e., source and destination IP addresses, source and destination port numbers, and protocol type). We explicitly distinguish *sessions* from *flows*: a session is bidirectional with a pair of unidirectional flows. A session begins with the connection initiation with the SYN packet and lasts until seeing the corresponding FIN packets for TCP. Since UDP has no connection establishment and termination, we assume that a session begins when the first packet appears and terminates when no corresponding packet that has the identical five tuples is seen for 60 seconds.

Among the large set of applications, we selected ten applications for our experiments as shown in Table I. There is no preference in the selection of applications but we simple chose applications that have the greater number of flows. We next discuss the experimental procedures we have conducted for evaluation.

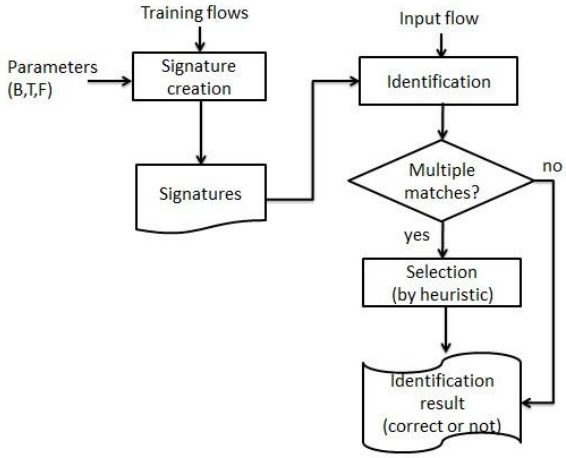

Fig. 4. The experimental procedure

B. The Experimental Procedure

Fig. 4 illustrates the overall procedure. The first stage in the procedure is to construct the signature set from the training data for all applications. In our experiments, we employed LCS because of its rich set of signatures

compared to LASER and AutoSig. In addition, unlike our expectation, the quality of LCS signatures is not inferior at all compared to the other techniques as also shown in Fig. 2. Anyhow, the purpose of our experiments is not on the discovery of the best way to search for adequate signatures but on helping identify flows effectively.

When constructing the signature set from the training flows, we employed three parameters to control:

- *Number of bytes examined (B)*: *B* is a byte size constraint that determines how many bytes of the flow data that will be used in searching for signatures. Since protocol specifications are often placed at the first part of a flow, limiting the number of bytes for inspection would be desirable for resource saving and early application identification [1] [4].
- *Minimal length threshold factor (T)*: *T* defines a character length constraint, based on which the signature that contains at least a certain number of bytes will be accepted. The minimal length threshold is computed by $\lfloor B * T \rfloor$. For example, if *B*=64 and *T*=0.1, the minimal length required for a signature is 6 bytes.
- *Coverage threshold factor (F)*: The coverage constraint *F* is used to keep track how frequently the signature is found for that application. Thus, it is simply calculated by dividing the number of flows the signature is observed by the total number of flows for that application in the training data set. If the fraction is greater than or equal to *F*, the signature will be considered as a candidate signature.

These parameters would dominantly work during the signature construction. The default parameters we used are: *B* = 64 (bytes), *T* = 0.1, and *F* = 10%. In other words, the first 64 bytes are examined in the signature construction and application identification stages, the minimal signature length is 6 bytes, and at least 10% of the total flows should contain the candidate sequence to be considered as a signature.

```
1   function construct_sig_table()
2   input: vector<Flow> flows;
3   output: table<sig,frac>sigTable;
4   begin
5   table<sig, bitmap>sigBitmap;
6   for(i = 0; i<flows.size(); i++) {
7   for(j = i+1; j <flows.size(), j++) {
8   Vector<sig> sigs = LCS(flows[i], flows[j]));
9   foreach sig (sigs) {
10  if (sig.length< threshold) continue
11  if (! sigBitmap.exists(sig)) {
12  create a new entry for sig and add it to sigBitmap;
13  }
14  sigBitmap.get(sig).bitmap.set(i);
15  sigBitmap.get(sig).bitmap.set(j);
16  }
17  }
18   }
19  foreachbmap (sigBitmap) {
20  sigTable.insert(bmap.key,bmap.bitmap.count/flows.size());
21  }
22  end
```

Fig. 5. The pseudo code for signature table construction

Fig. 5 provides a pseudo code for constructing a table that stores signatures for an application. The function takes a set of training flows to search for signatures, and produces a signature table (i.e., sig Table). In line 5-17, a bitmap is created for every signature found, if the size of the signature meets the minimal length constraint. The bitmaps will be used for computing "coverage" of signatures in the output table, as shown in line 19-23. Afterwards the fractions will be checked to see if they meet the coverage constraint by using the computed coverage fraction. Each table outputted by the function will hold the signatures for each application.

Once all the signatures are gathered, they will be inputted as a parameter for identification with an input flow in question. In the identification stage, each signature in the signature table will run through the LCS function with the input flow, and if the LCS function returns the signature itself, then the application signature is marked as identifying the flow as its application. If only one signature identifies the flow, it is a single match; if more than a single signature identifies the flow, then it should be a multiple match. Referring back to Fig. 4, if the flow has multiple matches, it will be examined by a selection heuristic to make the final decision.

For multiple matches, we set up the heuristics described in the previous section. In addition, we also used a simple heuristic based on a random function (RAND) that chooses one out of the candidate application set uniformly. Each heuristic chooses one application from the given candidate set, and the selected result is compared with the "real" application to which the input flow actually belongs.

We compared the selection heuristics to see the identification performance. The next section will describe the metrics employed for performance comparison.

C. Performance Metrics

To measure performance, we consider metrics widely adopted for application identification. Basically, *TP* stands for true positive that is the total number of flows correctly identified for the application, *FP* is false positive that is the total number of flows misclassified into the application, and *FN* is false negative that is the total number of flows that are not classified into the application (although they actually belong to the application). In particular, we consider the following measures that combine the basic metrics:

- *Precision*: the number of correctly classified flows out of the total number of flows classified into the application (i.e., *Precision = TP/(TP+FP)*).
- *Recall*: the number of correctly classified flows out of the total number of flows in the set (i.e., *Recall = TP/(TP+FN)*).

Hence, each of these measures is located between 0–100%; the greater, the better with respect to identification performance.

VI. Experimental Results

In this section, we present our experimental results and core observations. We first run experiments with the default parameters, and then examine sensitivity by exploring a diverse set of values for each parameter used for the experiments.

A. Results with the Default Parameters

The first run of the experiment uses the default parameters (i.e., B = 64, T = 0.1, and F = 10%). Our experiments in this section focus on the performance of the selection heuristics, particularly with respect to precision and recall.

Fig. 6 illustrates precision when we use the default parameters. While our heuristics show 81-94% on average, RAND works poorly with 74% for precision. *This strongly suggests that efficient handling of multiple matches is essential to achieve high identification* accuracy. PROB works fairly well with an average precision over 90%. Interestingly, as can be seen from the figure, GREATEST works very well with 94% precision on average despite its nature of simplicity. FRAC outperforms RAND, but it is pretty behind GREATEST and PROB with an average precision of 81%.

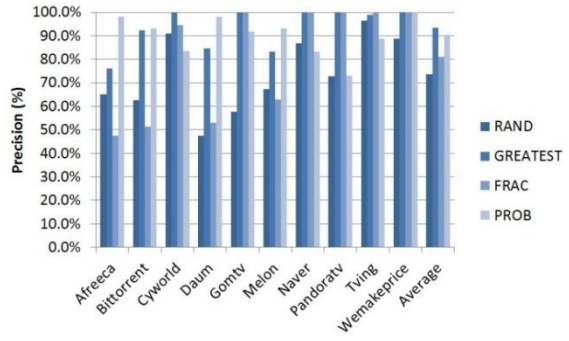

Fig. 6. Precision with default parameters

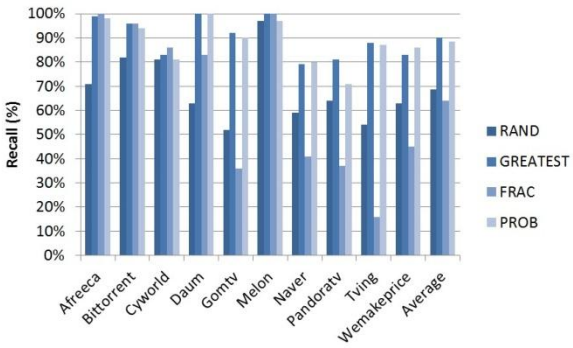

Fig. 7. Recall with default parameters

The precision metric considers false positive, while recall focuses more on false negative. Fig. 7 compares recall for the selection heuristics across the applications. The overall result is similar with the precisions in Fig. 6: GREATEST (90%) outperforms the other heuristics followed by PROB (88%) and RAND is the worst.

From the results shown in Fig. 6 and Fig. 7, we see that if many more signatures match with the input stream, there is a higher probability that the flow belongs to the application. To see if it is the case in another setting, we continue to explore a diverse set of values for the three parameters (i.e., B, T, and F).

B. Impact of the Number of Bytes Examined (B)

We next investigate the impact of the number of bytes inspected to the selection performance. Recall that the parameter B represents the number of bytes that are extracted from the flow data for both training and identification. We still fix T and F as the default values (i.e, T = 0.1, and F = 10%). In this experiment we used the following B values, B = {16, 32, 64, 128, 256}.

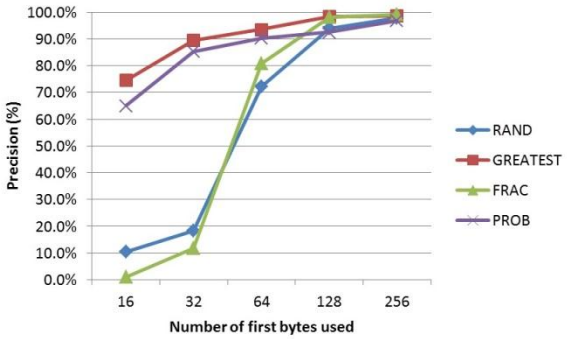

Fig. 8. Precision over the number of bytes inspected (B)

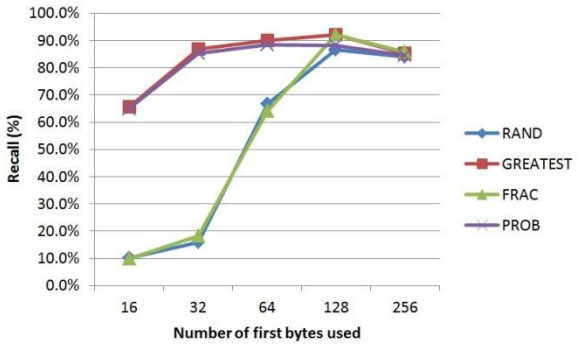

Fig. 9. Recall over the number of bytes inspected (B)

Fig. 8 and Fig. 9 show average precision and recall respectively over the number of bytes inspected (B). As can be seen from Figure 8, B has a significant impact to precision: as the number of bytes inspected increases, we can see precision also increases, implying greater identification accuracy. As in the default setting discussed in the previous section, GREATEST consistently outperforms the other heuristics over a diverse set of B. With B=128 (bytes), GREATEST shows a precision over 98%. However, we observed no significant difference with a greater B than 128 bytes. PROB follows GREATEST and shows over 90% precision with $B \geq 64$, and RAND is the worst.

One interesting observation is that FRAC significantly improves precision as B increases. In particular, FRAC is comparable with GREATEST with B=128 and slightly better when B=256 (98.7% for GREATEST and 99.1% for FRAC). In Fig. 9, FRAC also yields comparable results for recall when B≥128 bytes.

Fig. 9 shows recall over a diverse set of *B*. Both precision and recall show a similar pattern. However, recall does not make an improvement and is even degraded when *B*>128 for the entire heuristics. This suggests that using the first 128 bytes in a flow or smaller than that would be recommended for the signature-based identification.

The reason for the steady improvement with a greater *B* can be explained with the increase of the fraction of single match as *B* increases, as discussed in Figure 1. We observed that the true positive rate for single match is very promising over 98%. *This implies that with an adequate means to settle multiple matches, the signature-based identification would yield a high degree of classification accuracy.*

C. Impact of the Minimal Signature Lengththreshold (T)

We next investigate how the minimal signature length impacts the performance by varying *T*. Note that the minimal length to be accepted as a signature is computed by $\lfloor B * T \rfloor$. We used *T*= {0.05, 0.1, 0.15, 0.2, 0.25}, *B*=64, and *F*=10% in this experiment.

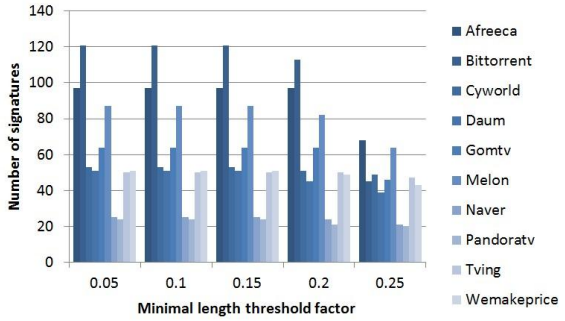

Fig. 10. The number of signatures with respect to the minimal signature length threshold factor (*T*)

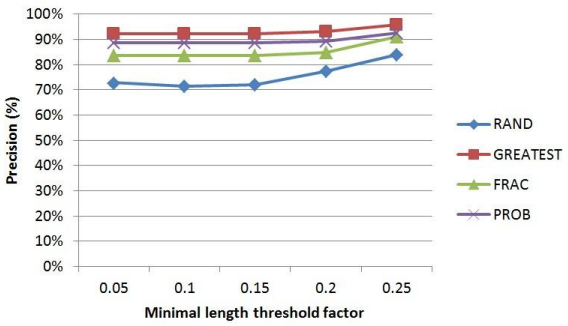

Fig. 11. Precision over the minimal signature length threshold factor (*T*)

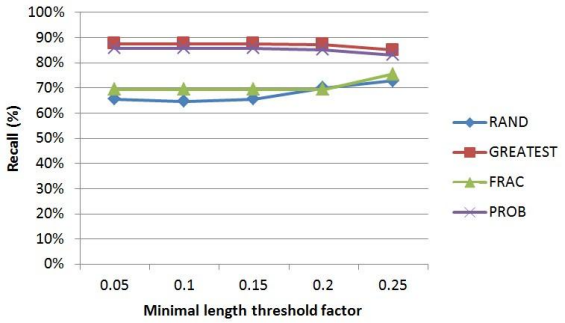

Fig. 12. Recall over the minimal signature length threshold factor (*T*)

Fig. 10 shows the number of signatures generated with different *T*. We can see the smaller number of signatures with a greater *T*, as expected. The average number of signatures is 62 with *T*=0.05, 0.1, and 0.15. It is reduced to 60 with *T*=0.2, and 44 with *T*=0.25.

As shown in Fig. 11 and Fig. 12. , a greater *T* slightly increases precision but decreases recall. This means that using a greater minimal length threshold can reduce false positive, but it increases false negative at the same time. In sum, there is a trade-off in choosing a value for the minimal signature length threshold with respect to the false positive and false negative rates.

D. Impact of the Coverage Factor (F)

The last experiment investigates the impact of coverage factor by varying *F*. We used *F* = {5%, 10%, 15%, 20%, 25%} and the default values for *B* and *T*.

As in Fig. 13, the greater coverage leads to the smaller number of signatures because a pattern should be more widely observed to be considered as a signature with a greater coverage factor. For example, with *F*=25%, any signature should appear in ¼ of the flows in the training data set, while *F*=5% requires any signature to be appeared in 1/20 of the training flows. For *F*=5%, the average number of signatures per application is around 180, while it drops to 16 for *F*=20%.

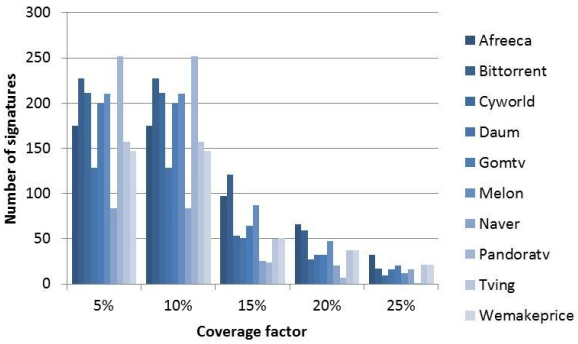

Fig. 13. The number of signatures with respect to the coverage factor (*F*)

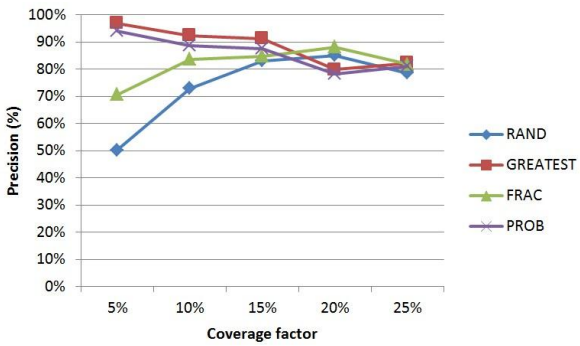

Fig. 14. Precision with respect to the coverage factor (*F*)

Overall, identification performance becomes worse as *F* increases. As shown in Fig. 14 and Fig. 15. both precision and recall decrease for GREATEST and PROB significantly over *F*. However, FRAC and RAND work better with *F* between 10% and 20%. GREATEST yields the best result with *F*=5%: 97% for both precision and recall. Therefore, it would be beneficial to choose a

relatively small value for the coverage factor around 5% of the entire number of flows in the training data set to expect better identification performance.

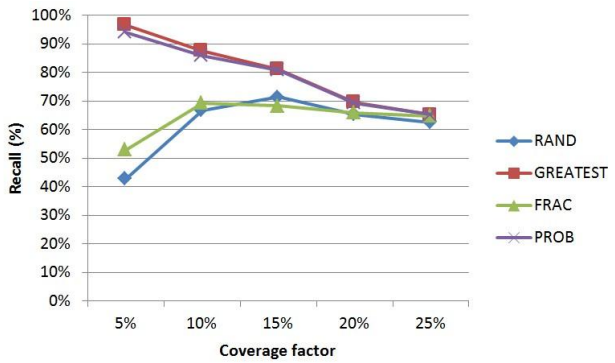

Fig. 15. Recall with respect to the coverage factor (*F*)

E. Discussion

We have seen that GREATEST outperforms the other selection techniques over diverse settings despite its nature of simplicity. To see why, let us take a look at some examples. From an Afreeca flow, we observed that 7Afreeca signatures identified it, 2 Pandora TV signatures did, and nothing for the other application signatures. Thus, GREATEST could correctly classify the flow into Afreeca. Here is another example for a Pandora TV flow: 2 Afreeca signatures matched, 2 for Daum, and 3 for Pandora TV, and as a result, GREATEST also identifies its application accurately.

The results for FRAC method show a different story. FRAC method is heavily influenced by the number of signatures generated for each application. Let us consider the two flows explained above again. For the Pandora TV flow, the fraction was 0.018 for Afreeca signatures, 0.034 for Daum, and 0.097 for Pandora TV. Even though Afreeca and Daum were only down by 1 in the GREATEST, they were at a disadvantage in FRAC, since Afreeca has 109 signatures, Daum has 59 signatures, and Pandora TV had 31 signatures. Therefore, FRAC was able to correctly identify the Pandora TV flow. However, if we take the second Afreeca flow example, the results gave Afreeca signatures a 0.064 and Pandora TV a 0.65. Even though Afreeca had the greatest number of matches, it lost to Pandora TV because it could produce a bigger fraction. The second example gives a look at why FRAC did not do as well as GREATEST in terms of accuracy.

Now we consider PROB with the above example flows. For the Pandora TV flow example, PROB had confidence P=0.23 for Afreeca, P=0.28 for Daum, and P=0.31 for Pandora TV. Even though not all the signatures for Pandora TV identified the flow, this example shows that the signatures that identified it had a relatively high confidence value. For the Afreeca flow, Afreeca had P=0.78 and Pandora TV had P=0.23, which enables PROB to classify the input flow accurately. Although PROB is slightly behind GREATEST with respect to overall accuracies, it would be rather reliable if there can be a high degree of discrepancies in terms of the available number of signatures for different applications, since PROByields stable accuracies fairly close to GREATEST.

VII. CONCLUDING REMARKS

While the signature-based approach is attractive with greater accuracy and thus adopted widely for network traffic classification, one potential challenge is multiple matches occurred in the identification process. Thus, it is necessary to effectively handle such multiple matches in order to obtain an adequate identification decision. To this end, we developed a set of selection heuristics that help battle multiple matches and help identify an input stream: GREATEST that selects based on the number of matches for each application, FRAC that considers the signature set size in addition to the number of matches, and PROB based on a probability that any of the matched signatures for an application appears in a flow belonging to that application.

To see effectiveness of the heuristic techniques, we conducted an extensive set of experiments with a recently collected data set. Our experimental results show that GREATEST and PROB fairly work well yielding up to 99%with respect to precision and recall. In particular, we observed that GREATEST consistently outperforms the other techniques despite its nature of simplicity.

There are some tracks needed to further be explored. From a set of experiments, we observed that LCS produces a rich set of signatures. However, we also observed that a substantial number of signatures for an application look closely similar. Refining the signature set would be beneficial not only for reducing the number of signatures but also for enhancing the quality of signatures. We plan to incorporate the refinement with the reconciliation presented in this paper, in order to provide a systematic methodology for the signature-based application identification. We also plan to employ additional traffic data sets for further verification of our techniques.

ACKNOWLEDGMENT

This work was supported by the IT R&D program of MOTIE/KEIT [10041548, “240Gbps real-time automatic signature generation system for application traffic classification supporting over 95% completeness and accuracy”]

REFERENCES

[1] L. Bernaille, R. Teixeira, and K. Salamatian, "Early application identification," in *Proc. ACM CoNEXT Conference*, 2006.

[2] L. Grimaudo, M. Mellia, and E. Baralis, "Hierarchical learning for," in *IWCMC*, 2012.

[3] T. E. Najjary, G. U. Keller, and M. Pietrzyk, "Application-based feature selection for internet traffic classification," in *Proc. 22nd International Teletraffic Congress*, 2010.

[4] G. Xie, M. Iliofotou, R. Keralapura, M. Faloutsos, and A. Nucci, "Subflow: Towards practical flow-level traffic classification," in *INFOCOM*, 2012.

[5] The BitTorrent Protocol Specification. (2008). [Online]. Available: http://www.bittorrent.org/beps/bep_0003.html,.
[6] M. Pietrzyk, T. E. Najjary, G. U. Keller, and J.-L. Costeux, "Hybrid traffic identification," *Technical Report EURECOM+3075*, 2010.
[7] L. Bergroth, H. Hakonen, and T. Raita, "A survey of longest common subsequence algorithms," in *Proc. Seventh International Symposium on String Processing Information Retrieval*, 2000.
[8] B.-C. Park, Y. J. Won, M.-S. Kim, and J. W. Hong, "Towards automated application signature generation for traffic identification," in *NOMS*, 2008.
[9] M. Ye, K. Xu, J. Wu, and H. Po, "AutoSig: Automatically generating signatures for applications," in *CIT*, 2009.
[10] J. Yang, Y. Xu, and Y. Shang, "An ecient parallel algorithm for longest," in *World Congress on Engineering* , 2010.
[11] J. Duan, R. Li, and Y. Hu, "A bio-inspired application of natural language processing," *Expert Syst. Appl.*, vol. 36, no. 3, pp. 4876-4883, 2009.
[12] S. Coull, J. Branch, B. Szymanski, and E. Breimer, "Intrusion detection: A bioinformatics approach," in *Proc. 19th Annual Computer Security Applications Conference*, 2003.
[13] Y. Wang, Y. Xiang, W. Zhou, and S. Yu, "Generating regular expression signatures for network traffic classification in trusted network management," *J. Netw. Comput. Appl.*, vol. 35, no. 3, pp. 992-1000, 2012.
[14] P. Haffner, S. Sen, O. Spatscheck, and D. Wang, "ACAS: Automated construction of application signatures," in *Proc. ACM SIGCOMM Workshop on Mining Network Data*, 2005.
[15] G. Szabó, Z. Turányi, L. Toka, S. Molnár, and A. Santos, "Automatic protocol signature generation framework for deep packet inspection," in *5th International ICST Conference on Performance Evaluation Methodologies and Tools*, 2011.
[16] E. Hjelmvik and W. John, "Statistical protocol identification with SPID: Preliminary results," in *6th Swedish National Computer*, 2009.
[17] Shark for Root. [Online]. Available: https://play.google.com/store/apps/details?id=lv.n3o.shark&hl=en
[18] Libpcap File Format. [Online]. Available: http://wiki.wireshark.org/Development/LibpcapFileFormat

Justin Tharp was born on May, 23 1988. Tharp worked on my Bachelor of Science at Texas A&M University and my major was Aerospace Engineering. Tharp is currently working on my Master's of Science at Texas A&M University – Commerce and my major is in Computer Science. His current job is GRADUATE ASSISTANT IN RESEARCH at Texas A&M University – Commerce.

Dr. Jinoh Kim received his Ph.D. degree in Computer Science from University of Minnesota, Twin Cities. He is currently an Assistant Professor in the Department of Computer Science at Texas A&M University-Commerce. The areas of research interests span from systems and networks, particularly including distributed systems, big-data computing, energy-proportional computing, and computer networks and security. Prior to that, he was a postdoctoral researcher at the Lawrence Berkeley National Laboratory for 2010-2011 and an Assistant Professor of Computer Science at Lock Haven University of Pennsylvania for 2011-2012. From 1991 to 2005, he was a researcher and a senior researcher at ETRI, Korea, and participated in various projects in the areas of network security and ATM Networks.

Dr. Sang C. Suh is currently a Professor and Head of Computer Science Department at Texas A&M University – Commerce, U.S.A. He founded a transdisciplinary research center called Intelligent Cyberspace Engineering Lab (ICEL) to launch and carry out many mega-scale transdisciplinary systems R&D projects. His research theme spans around many interdisciplinary research topics including computer science & engineering, systems science & engineering, biotechnology & bioinformatics, internet technology, and data mining, and knowledge engineering. The major research thrusts of his ICEL include design and modeling of intelligent systems with focus on Bio-informatics and biological sciences, knowledge discovery and representation, human computer interaction, ontological semantics on natural language processing, adaptive search engine for optimal knowledge discovery, and data mining.

Dr. Suh has authored and published over a hundred peer-reviewed scientific articles, several book chapters and books in the areas of data mining, bioinformatics, knowledge and data engineering, artificial neural networks, and adaptive search engines. He has chaired numerous technical sessions in many international conferences and served in the program committees of over twenty international conferences. Among his honors are Distinguished Service Award from World Congress - WorldComp, Distinguished Service Award for the Society of Design and Process Science, Texas Instruments Research Fellowship Award, Software Engineering Research Fellowship Award, and Minnie Stevens Piper Award nomination.

Dr. Suh has over 20 years of higher education teaching and research experience in computer programming, data mining, database systems, computer operating systems, automata and formal language theory, network programming, artificial intelligence, data warehousing, systems design and analysis, and expert systems. He currently serves as President of the Society for Design and Process Science which is a professional organization with focus on transdisciplinary research.

Hyeonkoo Cho received the B.E. and M.E. degrees in Information Communications Engineering from Chungnam National University in 1999 and 2001, respectively. During 2001-2009, he stayed in Virtual I Tech. Inc. and GTOPIA Inc. to research image communication, parallel processing, spatial image processing and GIS. After three year stay at Electronics and Telecommunications Research Institute (ETRI) in Korea to research network security and security situational awareness. Currently, He has been a senior principal member of engineering staff at Sysmate Inc. since 2012. His research interest includesimage processing, network security, DPI and network management.