Towards Automated Application Signature Generation for Traffic Identification

Byung-Chul Park , Young J. Won
Dept. of Computer Science and Engineering, POSTECH
Pohang, Korea
{fates, yjwon}@postech.ac.kr

Myung-Sup Kim
Dept. of Computer and Information Science, Korea University
Jochiwon, Korea
tmskim@korea.ac.kr

James W. Hong
Dept. of Computer Science and Engineering, POSTECH
Pohang, Korea
jwkhong@postech.ac.kr

Abstract— **Traditionally, Internet applications have been identified by using predefined well-known ports with questionable accuracy. An alternative approach, application-layer signature mapping, involves the exhaustive search of reliable signatures but with more promising accuracy. With a prior protocol knowledge, the signature generation can guarantee a high accuracy. As more applications use proprietary protocols, it becomes increasingly difficult to obtain an accurate signature while avoiding time-consuming and manual signature generation process. This paper proposes an automated approach for generating application-level signature, the LASER algorithm, that does not need to be preceded by an analysis of application protocols. We show that our approach is as accurate and efficient as the approach that uses preceding application protocol analysis.**

Keywords – Internet application traffic identification, traffic measurement and analysis, application signautre, automated signature generation

I. Introduction

A naive Internet application traffic identification technique, such as port-based identification, does not guarantee its accuracy due to the diversity of today's Internet traffic. Web storage (or Internet storage) and peer-to-peer (P2P) file sharing applications have rapidly grown in popularity, and their traffic occupies a great portion of the total Internet traffic volume [1]. In particular, a newer generation of P2P applications is incorporated with various firewall-bypassing strategies, such as ephemeral port allocation and relay node, to avoid detection and filtering. Yet, the port-based method is still a popular solution at the Internet backbone because of high volume of traffic and limited computing resource for traffic identification. This method yields very poor, inaccurate application identification results.

Signature-based identification has been proposed to remove any uncertainty of previous identification methodologies [2][8]. A signature is a portion of payload data that is static and distinguishable for applications, which can be described as a sequence of strings or hex values. Such approach is not new to the Internet worm research community (e.g., intrusion detection systems); however, their focus is limited to identifying the security threatening traffic only. Meanwhile, we focus on the application identification among innocuous traffic which widens the target traffic to be identified. Signatures have been manually extracted by the network administrators or security experts. It requires preceding protocol semantic analysis or empirical packet payload inspection for pattern recognitions. Human decision in such process causes a slow response time to deal with new applications. The quality of signatures also varies due to the level of expertise. It is an issue to find feasible signatures which delivers acceptable accuracy against the traffic in asymmetric routing environment (e.g., ISP's backbone links). Thus, we need a systematic approach to define and extract the signature for Internet application identification. It is an important step toward the traffic identification system with self-updating engine.

In this paper, we propose a LCS-based (Longest common subsequence) Application Signature ExtRaction algorithm (LASER), which can automatically determine a trustworthy pattern in the packet's payload without a prior knowledge of protocol formats. We evaluate the accuracy by measuring the closeness with the pre-discovered signatures. Although there have been a few research on worm signature generation, it is difficult to adopt the popular sliding window algorithm [7] that has been applied to worm signature generation due to the differences in traffic nature between innocuous network-based applications and worms. To our knowledge, no other research has attempted to automatically generate signatures for non-threatening Internet applications.

The remainder of this paper is organized as follows. Section II explains previous research on signature discovery and traffic identification. In Section III, we show the commonly used signature formats for worms and redefine the signature formats prior to describing our approach. In Section IV, we describe the details of the LASER algorithm used in the automated signature generation. Section V presents a validation on our proposed algorithm. Finally, we summarize our work and discuss some possible future work in Section VI.

II. Related Work

Some previous studies have shown their own P2P traffic identification method using signatures. Gummadi *et al.* [2] have used the application signature for the KaZaA workload characterization. Despite their efforts, the fundamental question of identification accuracy remains due to multiple signature conflicts. Sen *et al.* [1] have generated the signatures of a few P2P applications by analyzing the application-layer protocols. The protocol semantic analysis could improve the accuracy of signatures, but it causes poor efficiency in signature discovery and is not suitable for real-time analysis of the backbone traffic. P2P applications are the popular choice of target applications in other similar research because of its high traffic usage, complexity of traffic dynamics, and communication via undisclosed proprietary protocols. Obtaining the signatures of traditional applications, such as HTTP, FTP, and more, is relatively easy while their protocol format and communication behavior are publicly available. Overall, the signatures here are manually determined by using off-line analysis tools. They are also more suitable for the edge level traffic rather than the backbone because the underlying assumption is that every packet is being inspected.

Signature-based traffic identification has been widely adopted for intrusion detection systems to early detect and block worms [4][5][6][7]. These studies describe the development of automatic worm signature generation systems, which generate the signatures of anomaly traffic, such as port scanning and worm infection. It is their first priority to acknowledge even a slight possibility of vulnerability in the worm body. In this domain, a particular single byte or hexa-code representation is sometimes acceptable to recognize an incident like buffer overflow. Another difference to our study is that common vulnerability signatures, again like buffer overflow, can be sharable between the worms while the signatures for normal application must be unique. In fact, their definition of worm signatures is somewhat looser than the one in this paper because their goal is to discover as many flows as possible which are related to any kind of worm traffic behaviors. Short string or hexa sequences may not be specific enough to reduce the false positives for innocuous traffic identification.

In general, worm signature generation studies rely on a few variations of sliding-window algorithm with its particular own break points. Scheirer *et al.* [7] explained the types of available sliding-window algorithms and the selection of break points, which is the hex value of the instruction code corresponding to a specific action among worm's common behavior. The algorithms are called, Fixed Partition Sliding Window Scheme (FPSW), Variable-length Partition Sliding Window Scheme (VPSW), and Variable-length Partition with Multiple Breakmarks (VPMB). The widow is sliding across the multiple byte streams (or packet payloads) until they find the matching sequences. The problem occurs when applying these algorithms to normal traffic; there exist no such break points in the payload due to difference of traffic nature. We cannot tell where to stop and decide the appropriate comparison window size to start with. Therefore, it is difficult to apply the sliding window algorithm using break points to the general Internet applications such as P2P. This fact separates our work from the previous worm signature generation research.

In some limited instances, the signature can be extracted from the encrypted traffic. Ehlert *et al* [15] detected the hexadecimal patterns in the Skype (v.1.5, v.2.0) packet traces during the initial communication setup phase. Bernaille *et al* [16] indicated early signature signs for mixed application traffic which are carried over an encrypted SSL connection. Thus, finding a pattern in packet's payload is a valid approach even for encrypted traffic especially when it occurs in the early stage of connection.

III. Signature Formats

A. *Worm Signature Formats*

Newsome *et al.* [13] categorized the signature format for polymorphic worms, which refer to the worms that vary their payload on every infection attempt. Their definition does not completely cover the traffic other than worms; these signatures are simply sequences of substrings. The followings are the signature classification for polymorphic worm:

- *Conjunction signature:* A signature that consists of a set of substrings (or tokens), and matches a payload if all tokens in the set are found in it, in any order.
- *Token-subsequence signature:* A signature that consists of an ordered set of tokens.
- *Bayes signature:* A signature that consists of a set of tokens, each of which is associated with a score, and an overall threshold.

Brumley *et al.* [12] also proposed several representations of worm signatures: Turing machine signatures, symbolic constraint signatures, and regular expression signatures. However, there exists tradeoff between the expression power and matching efficiency. It is not suitable for real-world applications, especially when matching against large number of application signatures.

B. *Innocuous Application Signature Formats*

Signature-based identification research that proceeded up to now defined their own signature formats and developed solutions corresponding to their formats to analyze application traffic. We categorize several established signature formats as follows:

- *Common string with fixed offset:* defines the signature as string or hex values that appear on a specified offset of packet payload. In most cases, the offset is the beginning of packet's payload (e.g., eDonkey '0xe3' at offset 0)
- *Common string with variable offset:* this format is almost the same as the former except for the offset value. In common string with fixed offset, the offset of signature is constant; however in this case, common string can appear at any position in payload.
- *Sequence of common substrings:* different from the former two formats, it defines the signature as an order of some substrings that appear on the payload. It means that one string or hex value cannot be a signature, since when one string is considered as a signature, the accuracy cannot be guaranteed.
- *Behavioral signature:* depends on statistical characteristic of application traffic such as packet inter-arrival time, minimum packet size, maximum packet size, etc. Machine learning techniques based on various measurement values are widely studied. This format was not considered in our work since our goal is to achieve automated signature generation by inspecting packet data.

Karagiannis *et al.* [8] used the common string with fixed offset for their work. However, when used for signature generation, the accuracy of signature can decrease, since the signature is generated without an analysis of application protocols. For example, if the protocol field's length is variable, then the offset is variable as well. Thus, such problem is solved by using common string with variable string format. Since many applications use HTTP protocol for convenience these days, 'GET' or 'HTTP' string on payload are frequently found. However, those strings cannot distinguish applications; hence for the signature formats mentioned prior to this, those strings can not have any meaning as a signature. Sen *et al.* [2] described how to increase the signature accuracy by using a sequence of common substrings format. If we use this format, 'HTTP' or 'GET' can be treated as a part of signature.

Taken altogether, the sequence of common substrings format is a superset and the most accurate format among all signature formats except for the behavioral signature. Therefore, our approach uses a sequence of common substrings.

IV. Signature Generation Using the LASER Algorithm

Our automated signature generation consists of two parts: 1) sanitized packet collection and 2) signature (or pattern) extraction. The sanitized raw packets here refer to the packets belonging to the target application only. We have developed a continuous packet dump agent using Winpcap [10] to collect the packet trace for every running process in the OS. The collecting agent divides the sanitized packets according to each flow and store them in a separate packet dump file tagging with the origin process name. Keeping the data sets separate according to each flow and the process name is important in order to reduce unnecessary packet comparison overhead in the upcoming signature extraction step. If the given data set is a mixture of many different applications, it may generate too many garbage values, such as meaningless single-byte match. Such design decision is necessary to guarantee the signature extraction efficiency and accuracy; thus, we remove any uncertainty of traffic being fed to the signature extraction algorithm.

For the signature extraction step, we adopted the LCS (Longest Common Subsequence) algorithm [11], which has been mainly used for DNA sequence matching in bioinformatics applications. DNA sequence matching calculates common subsequences and measures the similarity between DNA of two (or more) different organisms. The basic idea of extracting common strings from packet payloads is analogous to DNA sequence matching; however, there are fundamental differences between DNA and packet payload, such as a basic unit of string composition. Thus, we modified LCS algorithm suitable for signature generation. Packet payloads substitute for DNA sequence, and the LASER algorithm extracts common substrings as a signature from the traffic generated by the same application.

A. *Constaints*

Because the application signature generation deals with a larger number of strings being compared, some modifications on LCS are needed. We describe the following constraints as our modification to LCS. Note that, we are also interested in all possible common substrings that can be found along with the longest one.

Number of packets per flow: A flow is a collection of packets that share the identical source IP, destination IP, source port, destination port, and protocol. It would be virtually impossible and unnecessary to exploit all these packets, especially conducting an expensive deep packet inspection. Sen *et al.* [2] also observed that the concrete signature exists in the initial few packets of the flow.

Minimum substring length: The cost of signature matching is proportional to the length of the signatures [14]. The generated signature consists of the sequence of substrings. The length of these substrings reflects their significance as a reliable signature for accurate identification. To avoid any trivial signatures, a minimum bound for the substring length should be considered as a constraint for the modified LCS algorithm. For example, if a single character is determined as one of the common substrings during the signature generation process, it would be difficult to conclude that it was an actual signature unless it appears repeatedly in the fixed offset position. In fact, the applications that employ the HTTP protocol contain the frequent use of '/' in its packets. By having this length constraint, we prevent these single and multi-positioned characters from involving in the sequence of common strings.

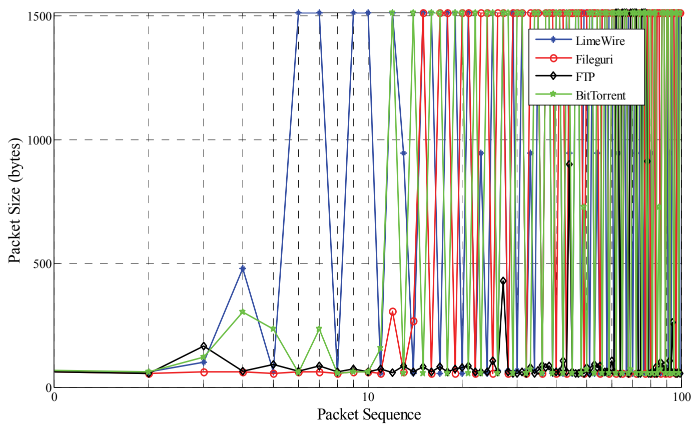

Figure 1. Packet size distribution of first one hundred packets when initializing the file download - (1) LimeWire, (2) Fileguri, (3) FTP, (4) BitTorrent

Packet size comparison: It increases the chance of finding a reliable signature if the packets are grouped in a similar purpose (e.g., signaling traffic, download traffic) and traffic characteristics. One of the fundamental characteristics for deciding the closeness between the packets is a packet size. Figure 1 illustrates the packet size of first 100 packets on each flow for four different applications. Due to space limitation, we present the byte distributions of these applications only. A first few packets are generated to initiate the connection handshake, such as login phase. The volume of transmitted data at the initial phase is relatively smaller than that of actual downloading. A concrete signature exists only in the initial few packets. Thus, the comparison between the small handshake packets and the large downloading packets is undesirable while generating a reliable signature. Consequently, the LCS signature generation relies on the comparison of packets within the similar range of packet sizes. For example, LimeWire, a popular Gnutella application, shows that its signaling traffic – connection control, search, etc. – are dealt with relatively light packets of average 390 bytes [17]. However, the average packet size for the traffic containing the actual download is 1460 bytes. These two types of traffic are less likely to share any signatures between them because their role is different in the application.

B. LASER Algorithm

Algorithm 1. Signature Generation Using LASER

```
1: procedure Signature_Generation ()
2:   Flow_Pool {$F_1$[]…$F_x$[]} $\leftarrow$ Santized_packet_collector
3:   $F_1$[] $\leftarrow$ Iterate, packet dump for Flow 1
4:   $F_2$[] $\leftarrow$ Iterate, packet dump for Flow 2
5:   while i from 0 to #_packet_constraint do
6:     while j from 0 to #_packet_constraint do
7:       if |$F_1$[i].packet_size - $F_2$[j].packet_size| < threshold
8:         result_LCS $\leftarrow$ LASER ($F_1$[i], $F_2$[j])
9:         LCS_Pool {} $\leftarrow$ Append result_LCS, end if
10:      j++, end while
11:  i++, end while
12:  S $\leftarrow$select the longest from LCS_Pool
13:  while i  from 0 to number of rest flows of Flow_Pool do
14:    $F_i$ $\leftarrow$ select one from the rest of Flow_Pool
15:    result_LCS $\leftarrow$ LASER (S, $F_i$)
16:    S $\leftarrow$ select the longest from result_LCS
17:    i++, end while, end while
18:  return S

19: procedure LASER (Packet$_A$[1...m], Packet$_B$[1...n])
20:  Packet$_A$ [m...1] $\leftarrow$ Reverse byte stream
21:  Packet$_B$ [n…1] $\leftarrow$ Reverse byte stream
22:  Matrix [m][n]
23:  while i from 0 to m do
24:    while j from 0 n do
25:      if i = 0 or j = 0, then Matrix [i][j] $\leftarrow$ 0
26:      else if Packet$_A$ [i] = Packet$_B$ [j], then
27:        Matrix [i][j] $\leftarrow$ ‘Diagonal’
28:     else if Matrix[i][j] != p[i][j-1], then
29:        Matrix[i][j] $\leftarrow$ ‘Up’
30:     else Matrix[i][j] $\leftarrow$ ‘Left’, end while
31:  end while
32:  i $\leftarrow$ m-1; j $\leftarrow$ n-1                          //Tracking
33:  while Matrix[i][j] != 0 do
34:    if Matrix[i][j] = ‘Left’, then j--
35:      else if Matrix[i][j] = ‘Up’, then i--
36:      else if Matrix[i][j] = ‘Diagonal’, then do
37:        Substring $\leftarrow$ Append Packet$_A$[i]
38:        if Matrix[i-1][j-1] != ‘Diagonal’, then
39:          Substring $\leftarrow$ Append special break point character
              (e.g. ‘/’)
40:        i--; j--, end while
41:   while tokenizing substring based on break point do
42:     if token_length > minimum_substring_length_constraint
43:      then, result_LCS $\leftarrow$Append token_substring, end while
44: return result_LCS
```

Algorithm 1 describes the overall signature generation algorithm and the LASER algorithm. The target application traffic is aggregated to a flow which is a collection of packets sharing the identical 5-tuple information. Initially, the two distinct flows – F_1 and F_2 – are used as input to the first iteration. The inputs to the LASER algorithm itself are two distinct byte streams of packet payloads that belong to F_1 and F_2 (Line 8).

Figure 2 illustrates the step after the packet selection based on the packet size constraint (line 7) and the number of packets per flow constraint (line 5 and 6). It shows two byte streams from LimeWire. The byte stream from the sender uses Morpheus which is another descendant application of Gnutella protocol and accesses the same group of LimeWire servers. The candidate signature in the figure shows the preliminary signature as the following: ‘*HTTP 200 OK Server : * e * Content-type : * e **’

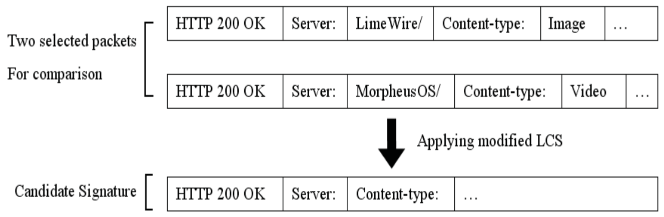

Figure 2. Applying the modified LCS algorithm to two independent LimeWire packet payloads

However, the substring ‘**e**’ is too short to have any meaningful decision. We remove this ambiguous substring from the current signature according to the minimum substring length constraint (line 42). The candidate signature after the first iteration is defective because its sample set is small – only two flows (up to line 12). More details on generating the final signature will be covered in the signature refinement process. In addition, the substring like, ‘HTTP 200 OK’ is often engaged by many other applications using HTTP protocol. Its lone usage is insufficient to be a unique signature. When it comes as a part of common sequence, it can be a meaningful substring.

C. Signature Refinement

This is a refining process of eliminating trivial strings (or hex) from the candidate signatures by iterating through the collected flows. Trivial strings imply any string that is not qualified for concrete signature, like replaceable information in the payload (e.g., IP address, object referrer in URL, etc.) In other words, the strings do not appear in the application protocol format.

Table I. Comparison of the generated signatures by packet analysis, protocol analysis, and LASER methods

	LimeWire	**BitTorrent**	**Fileguri**
Packet Analysis	"GNUT"	"0x13Bit"	"Freechal"
Protocol Analysis	'GET' or 'HTTP' followed "User-Agent: Limewire" or "UserAgent: Limewire" or "Server: Limewire"	"0x13BitTorrent protocol"	N/A
Automated Signature Generated by LASER	Sequence of 10 substrings "LimeWire" "Content-Type:" "Content-Length:" "X-Gnutella-Content-URN" "run:sha:1" "X-Alt" "X-Falt" "X-Create-Time:" "X-Features:" "X-Thex-URI"	Sequence of 1 substring "0x13BitTorrent protocol"	Sequence of 6 substrings "HTTP" "Freechal P2P" "User-Type:" "P2P-ErrorCode:" "Content-Length:" "Content-Type:" "Last-Modified"

The candidate signatures after the first iteration using the two flows are fed back to the LASER procedure along with another flow (line 12-17). A new set of substrings after each iteration is added to or removed from the pool of candidate signatures. The iteration continues until the numbers of candidate signatures are fixed or all the flows are iterated. Ultimately, as more training steps are processed, the signature accuracy increases. The signature refinement process can be simply expressed as follows:

```
Candidate_signature_1 = Signature (Flow 1, Flow 2)
Candidate_signature_2 = Signature (Flow 3, Candidate_signature_1)
…
Candidate signature_n = Signature (Flow n+1, Candidate_signature_n-1)

If Candidate_signature_n = Candidate signature_n-1
        For the certain iteration counts then
        Candidate_signature_n is the final signature
```

V. Validation

We have selected three popular P2P applications - LimeWire, BitTorrent, and Fileguri – based on their popularity and current availability of the concrete signatures to validate our LASER algorithm. LimeWire is well-known P2P application worldwide that uses the Gnutella protocol. BitTorrent is also a popular file sharing application for large files, such as movies. Both applications are popular choices for previous signature-based identification research [2][8]. Finally, Fileguri is a regional P2P application. Won *et al.* [9] presented the signature of this application.

A. Comparison with Manually Searched Signature

Table I presents the signatures generated by the following three distinct generation methodologies: Packet analysis, protocol analysis, and our proposed method. Packet analysis extracts a signature from raw packets by inspecting every individual packet without a prior knowledge of the application protocol structure and behavior [8][9]. Meanwhile, protocol analysis decides the signatures based on the corresponding protocol semantics which often requires protocol reverse engineering [3]. When the application protocol is publicly available, we assumed that protocol analysis is the most accurate method for signature generation. In fact, the previous research [4] claims that they achieve over 99% signature accuracy.

For the LASER algorithm, the following constraint conditions are applied. The number of initial packets to compare is set to 10 according to [2]. We also cross-verify this count by measuring packet size distribution in Section IV. The minimum substring length is set to 3. These constraints are adjustable depending on the type of applications.

LimeWire: The signature found after packet analysis is "GNUT" which is relatively short. Its sole use in traffic identification cannot guarantee the identification correctness due to the possible signature collision with non-LimeWire packets. The signatures by protocol analysis are compared to those by the LASER algorithm, which shows that they share one clear common substring – 'Limewire'. The substrings that appear only in the list of LASER signatures, such as "X-Gnutella-Content-URN," are clearly one of the Gnutella protocol's fields.

BitTorrent: The packet analyzed signature is shorter than the protocol analyzed signature, but the LASER signature finds the identical signature from protocol analysis.

Fileguri: This application uses undisclosed proprietary protocol; hence, protocol analyzed signature is currently unavailable. It is discovered that the LASER signatures are also a super set of the packet analyzed signatures, which is similar to the LimeWire's case.

We observe that the LASER signatures are either identical or close to the signatures from the rest of the methods. In fact, some sets of strings, which were spotted only in the LASER signature, were actually a part of the original application protocol format. This leads to the conclusion that the LASER algorithms can effectively extract the common subsequence strings. We claim that the accuracy of LASER signatures is guaranteed in the term of closeness of string appearance.

B. Accuracy Evaluation

In order to validate accuracy of the signatures generated by LASER, we check and confirm traffic classification result by applying the signatures to traffic captured from our campus backbone network.

a) Accuracy Evaluation Metrics

Here by, we define the following metrics to determine the signature accuracy.

- *False Positive (FP)*: Application signature erroneously classifies non-application traffic as the target application traffic.

$$FP = \frac{\text{Non-application traffic classified as application traffic}}{\text{Total application traffic}}$$

- *False Negative (FN)*: The given application signature is ineffective in detecting the traffic which belongs to the same target application.

$$FN = \frac{\text{Application traffic classified as non-application traffic}}{\text{Total application traffic}}$$

- *Overall Accuracy (OA):* OA measures not only the accuracy of a specific signature but also the accuracy of the entire classification result reflecting all signatures taken as a whole. This metric can be considered as a synthetic value that measures false positive/negative and true positive/negative as well as conflicts between each signature.

$$OA = \frac{\text{Total traffic - (Total FP traffic + Total FN traffic)}}{\text{Total traffic}}$$

The former two indicate undesirable misclassification ratios while the latter shows a desirable accuracy of the total classification result. Therefore, low false positive/negative and high overall accuracy are required to ensure that the signatures are accurate.

b) Data Set & Evaluation Methodology

We analyzed full packet trace collected from the POSTECH campus network. Figure 3 shows POSTECH's core network and its Internet junction where two 1G Ethernet links are connected to two separate ISPs. No port blocking policy is in effect; therefore, it results in the unrestricted amount of peer-to-peer traffic in the campus. The collected packet trace covers 3 hour period on August 16, 2007 and total traffic volume was about 450 Gbytes.

In order to measure the accuracy of traffic classification, it is crucial to obtain a firm ground truth for verification. A few previous studies in Section 2 rely on port-base traffic classification and exhaustive packet payload inspection as their methods to provide a comparable set of ground truth traffic. They also tend to depend on the traffic collected at the isolated and controllable environment, such as Virtual Private Network. It is convenient to assume that such traffic contain less complicated traffic (e.g. peer-to-peer traffic) because its traffic usage are generally restricted to the office related applications only. In fact, the reliability of these ground truth traffic information is questionable. The corresponding FP and FN values could contain marginal errors.

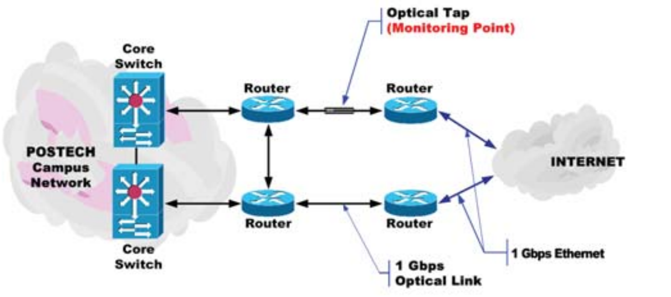

Figure 3. POSTECH Internet Junction

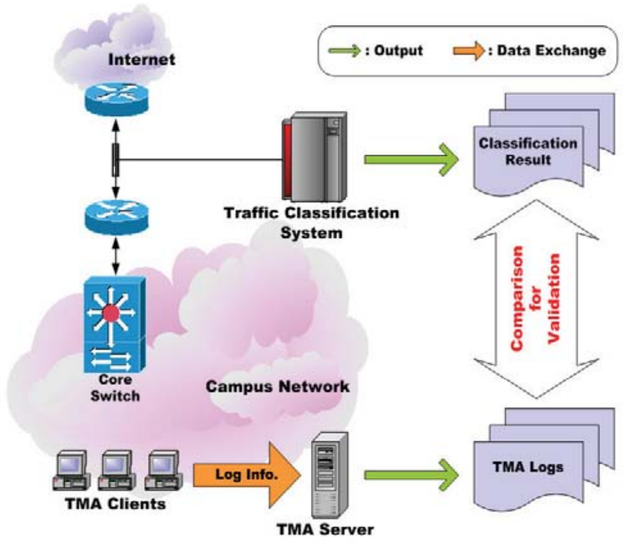

Figure 4. Validation using TMA

To obtain the absolute ground truth traffic and its application information, we focus on the origin of traffic itself. We deploy Traffic Measurement Agents (TMA) on the selected hosts in the network. TMA, a Windows based client program, monitors a designated network interface of the host and generates the traffic summary log of the interface. It also constantly monitors which process in the system is responsible for the packet being transmitted and received via the interface. The log is composed of lines of traffic summary according to each corresponding process which can be later used as the ground truth for FP and FN verification. Figure 4 shows the usage of TMA. The TMA log data are generated from campus network's edge client and transmitted to the TMA log sever. Then the server keeps the accumulated log data which can be used to determine the overall accuracy of the traffic being classified using the LASER signatures. This TMA approach can be applied to various other classification methods measuring accuracy as well.

c) Result

After installing TMA on two client computers, we operated the selected three applications as well as other Internet applications so that TMA leaves various log data on the client computers. At the same time, we capture the packet trace at the campus backbone. We conducted classification work by applying the corresponding LASER signatures on the traffic which belongs to the two TMA clients. Such traffic can be separated from the whole collected traffic traces by IP filtering. The reason why we separated the-client-causing traffic and the rest of packet traces, instead of directly collecting traffic from two TMA clients, involves asymmetric routing behavior at the backbone.

When passing through backbone network, each packet may be affected by reordering of packet sequence or asymmetric routing. Considering that backbone links are the points where most of the traffic classification systems are performing, the traffic gathered from end host can not reflect the changes done by the backbone network. The classification result is presented in table II.

Table II. Evaluation Result

Application	TMA Log (MB)	Classification Result (MB)	False Negative (%)	False Positive (%)
LimeWire	1223.36	1120.35	8.42	0
BitTorrent	4190.07	3754.30	10.40	0
Fileguri	3189.61	3177.17	0.39	0
Others	12482.69	13033.91	-	-
Total	21085.73MB		-	-
Overall Accuracy	97.39 (%)			

- *FP Analysis*: Due to the restricted nature of signature format that stems from the sequence of substrings, none of the traffic caused by other various applications is misclassified as the traffic generated by three main applications. It is worthwhile to note that a large amount of HTTP traffic generated by Internet Explorer was not classified as LimeWire or Fileguri that employs HTTP protocol. This shows that signature's unique sequence even with the existence of common substrings like 'HTTP' keeps the false positive ratio quite low.
- *FN Analysis*: All three application signatures had false negatives. It means that there exists application traffic which cannot be detected by the given signature. While examining the misclassified traffic separately, we made a few observations that might be accounted for such phenomenon. One of the observations was that some packets did not have any application level packet payload except for TCP/IP headers. Application signatures exist in application-level packet payloads; thus, flows generated under the circumstance where the application only exchanges the packets for TCP connection only containing TCP flags like SYN, SYN-ACK and ACK, cannot be classified as application traffic. The second reason was the packet loss due to asymmetric routing. Since our monitoring point is just one of the two Internet junctions of the campus network, there is a possibility of alternative routing path for the signature containing packets. Again, this is a common problem for backbone monitoring environment. At last, the web browser embedded in the application was responsible for FN. In this case, although the origin of traffic was non-HTTP application, the corresponding traffic was classified as pure web traffic.

While the overall accuracy was as high as 97.39%, false negative values of BitTorrent and LimeWire were nearly 10%. Compared to false negative figures from other research [3] which is 9.90% and 4.97% respectively, the resultant figure is still acceptable. Although the test cases were simple and limited, we could confirm that application signatures generated by the LASER algorithm have reasonable accuracy with low FN and FP. We will install more TMA clients to obtain more precise evaluation values as our future work.

C. *Efficiency of LASER Algorithm*

Figure 5 shows the signature refinement process as the number of iteration increases. It indicates a decrease in length of candidate signature as refining iteration continues. After the certain point, the length of candidate signatures remains still which implies an iteration termination point in the proposed algorithm. The number of refining iteration to have the final signature is below 10 on all three applications. Surprisingly, a small amount of traffic input is needed.

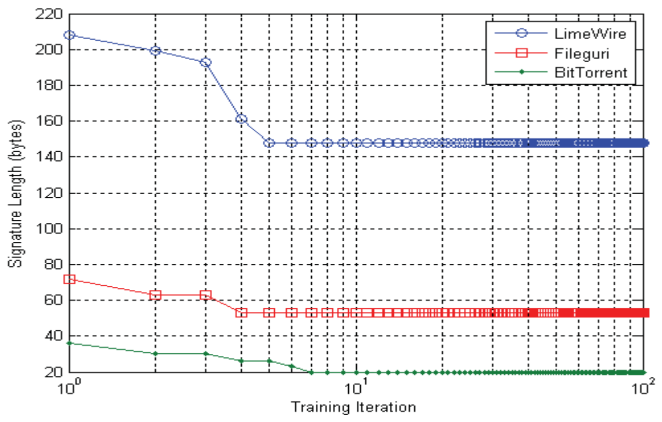

Figure 5. Signature refining iteration vs. Candidate signature length

Table III shows the other application signatures found by the proposed method. Verification on these signatures is not possible because the signatures by other research are either absent or out-dated (e.g., version update). Afreeca TV and PD-BOX are popular P2P-based personal broadcasting application and Web storage application in Asia, respectively. To analyze the effectiveness on encrypted payload, Skype and KaZaA are selected. We try to select at least one representative application from the recent emerging service categories, such as VoIP, P2P, broadcasting, encryption-enabled applications, and etc. Note that, the previously known signatures for Skype v1.5 and v2.0 [15] are not valid any more due the version change in Skype. The current version of Skype (v3.0) does not show any signs of patterns in the payload due to more enhanced encryption. A different type of signature measures, like packet size (e.g., 18 byte UDP packet by Skype), could be an alternative solution to this undetected case.

Table III. Other Application Signatures by LASER

	Automated Signature Generation by LASER
FTP	"230 logged"
Afreeca TV	"0x02 02 21 CB 4E 02 00 00 6D DB 00 00", "0x00 00 00 00", "0x7E 00 00"
PDBOX	"0x00 00 01 03 16 05 00 00 08 00 00 00 1E 05 01 03 00 00 00 00 32 00 00 00 57 37 59 5D"
Skype (v3.0)	No signature can be found
KaZaA(v3.25)	"HTTP1.1" "KazaaClient" "X-Kazaa-Username:" "X-Kazaa-Network:" "X-Kazaa-IP:" "X-Kazaa-SupernodeIP:" "X-Kazaa"

VI. Concluding Remarks

In this paper, we have introduced a new approach which automatically generates application signatures without a prior exhaustive search of application protocols for application traffic identification. We particularly focused on the string or hexa patterns in the payload which can be easily applied to identifying asymmetric Internet backbone traffic traces. The proposed method using the LASER algorithm overcomes the shortcomings of manual search for signatures and improves the signature accuracy. We also found that a small training data set is sufficient enough in finding reliable signatures. By comparing with the previously found signatures in other research, we observe that the LASER algorithm yields reliable signatures while minimizing human intervention. For future work, we plan to deploy multiple signature generation agents that collect the real-time training traffic trace and report the generated signatures periodically. We also plan to share the signature database within the measurement research community in order to verify further on the signature accuracy.

Acknowledgment

This research was supported in part by the Electrical and Computer Engineering Division at POSTECH under the BK21 program of Ministry of Education and the Program for the Training of Graduate Students in Regional Innovation of Ministry of Commerce, Industry and Energy of the Korean Government, and by the MIC(Ministry of Information and Communication), Korea, under the ITRC (Information Technology Research Center) support program supervised by the IITA(Institute of Information Technology Advancement)" (IITA-2007-(C1090-0701-0045))

References

[1] S. Sen and J. Wang. "Analyzing peer-to-peer traffic across large networks," 2002 ACM SIGCOMM Internet Measurement Workshop, Marseilles, France, Nov. 2002.

[2] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan. "Measurement, modeling, and analysis of a peer-to-peer File-sharing workload," 19th ACM Symposium on Operating Systems Principles (SOSP-19), October 2003.

[3] Subhabrata Sen, Oliver Spatscheck, Dongmei Wang. "Accurate, Scalable In-Network Identification of P2P Traffic Using Application Signatures," WWW 2004 Conference.

[4] H. Kim, B. Karp. "Autograph: Toward automated, distributed worm signature detection," 13th Usenix Security Symposium, 2004.

[5] S. Singh, C. Estan, G. Varghese, S. Savage. "Automated Worm Fingerprinting," 6th USENIX Symposium on Operating Systems Design and Implementation, 2004.

[6] Sumeet Singh, Cristian Estan, George Varghese and Stefan Savage. "The EarlyBird System for Real-time Detection of Unknown Worms", UCSD Tech Report CS2003-0761, August 2003.

[7] W. Scheirer, M. Chuah. "Comparison of Three Sliding-Window Based Worm Signature Generation Schemes," Technical Report LU-CSE-05-025.

[8] Thomas Karagiannis, Andre Broido, Michalis Faloutsos, and KC Claffy. "Transport layer identification of p2p traffic," Internet Measurement Conference (IMC), 2004.

[9] Young J. Won, Byung-Chul Park, Hong-Taek Ju, Myung-Sup Kim, and James W. Hong. "A Hybrid Approach for Accurate Application Traffic Identification," IEEE/IFIP E2EMON Workshop, Vancouver, April 2006, pp. 1-8.

[10] Winpcap. http://www.winpcap.org/.

[11] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, *Introduction to Algorithms*, 2nd Edition, MIT Press, 2001.

[12] David Brumley, James Newsome, Dawn Song, Hao Wang, and Somesh Jha. "Towards Automatic Generation of Vulnerability Signatures," IEEE Symposium on Security and Privacy, May 2006.

[13] James Newsome, Brad Karp, Dawn Song. "Polygraph: Automatic Signature Generation for Polymorphic Worms," IEEE Security and Privacy Symposium, May 2005.

[14] James Newsome and Dawn Song. "Dynamic Taint Analysis: Automatic Detection, Analysis, and Signature Generation of Exploit Attacks on Commodity Software," Network and Distributed Systems Security Symposium, Feb 2005.

[15] Sven Ehlert and Sandrine Petgang, "Analysis and Signature of Skype VoIP Session Traffic," Franunhofer FOKUS Technical Report NGNI-SKYPE-06b, Berlin, Germany, July, 2006.

[16] Laurent Bernaille and Renata Teixeira, "Early Recognition of Encrypted Applications," PAM 2007, Louvain-la-neuve, Belgium, April 5-6, 2007, pp. 165-175.

[17] The Gnutella protocol specification, http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf.