Received: 27 March 2014 | Revised: 30 March 2016 | Accepted: 12 October 2016

DOI 10.1002/nem.1959

WILEY

RESEARCH ARTICLE

Header signature maintenance for Internet traffic identification

Sung-Ho Yoon[1] | Jun-Sang Park[1] | Baraka D. Sija[1] | Mi-Jung Choi[2] | Myung-Sup Kim[1]

[1] Department of Computer and Information Science, Korea University, Sejong, Korea

[2] Department of Computer Science, Kangwon National University, Chuncheon, Korea

Correspondence
Myung-Sup Kim, Department of Computer and Information Science, Korea University, Sejong 339-700, Korea.
Email: tmskim@korea.ac.kr

Summary

Various traffic identification methods have been proposed with the focus on application-level traffic analysis. Header signature–based identification using the 3-tuple (Internet Protocol address, port number, and L4 protocol) within a packet header has garnered a lot of attention because it overcomes the limitations faced by the payload-based method, such as encryption, privacy concerns, and computational overhead. However, header signature–based identification does have a significant flaw in that the volume of header signatures increases rapidly over time as a number of applications emerge, evolve, and vanish. In this article, we propose an efficient method for header signature maintenance. Our approach automatically constructs header signatures for traffic identification and only retains the most significant signatures in the signature repository to save memory space and to improve matching speed. For the signature maintenance, we define a new metric, the so-called signature weight, that reflects its potential ability to identify traffic. Signature weight is periodically calculated and updated to adapt to the changes of network environment. We prove the feasibility of the proposed method by developing a prototype system and deploying it in a real operational network. Finally, we prove the superiority of our signature maintenance method through comparison analysis against other existing methods on the basis of various evaluation metrics.

1 | INTRODUCTION

As high-speed Internet has become more widespread and Internet-based applications more diverse, network management has become increasingly important. The availability of high-speed Internet has led users to demand more stable services with a guarantee for quality of service. From the perspective of Internet service providers and Internet contents providers, the need to provide a diverse and high-quality service has increased, whereas there remains pressure to minimize their capital expenditures and operating expenses. However, network burden is serious because of limited network resources and fast-growing traffic. To achieve efficient network operation and management, it is necessary to detect killer applications consuming high bandwidth and analyze their behavioral patterns. Because the performance of network policies that block or adjust target traffic is highly dependent on the identification results, application-level traffic identification should be preemptively accomplished.[1–4] Such identification results are also used in numerous areas of network management including traffic engineering, network planning, quality of service planning, and service-level agreement management.

The goal of application-level traffic identification is to accurately determine the application names such as Skype, uTorrent, and DropBox that generate network traffic. In the case of Web-based services, each service such as YouTube, Google, and Facebook is considered an individual application. Previously, various methods using a diverse set of traffic features have been suggested for traffic identification. However, all such methods encounter limitations when applied to a real operational network. Examples of such limitations include difficulty in signature creation, computational complexity, limitations related to privacy concerns, and real-time control problems.[5–7]

In particular, header signature (HS)–based traffic identification[8–10] has garnered much attention because it overcomes several limitations such as encryption, privacy concerns, and computational overhead that are faced by the payload-based method, which inspects payload in each packet to find a predefined string. The HS is the combination of the 3-tuple (Internet Protocol [IP] address, port number, and L4 protocol) of a server that a specific application uses to communicate with the end hosts involved. If a server provides a specific application or service for a long time, the server 3-tuple is extracted as an HS of the application and is stored in the signature repository. The HS is used to identify traffic by matching packet headers. Although there are several peer-to-peer (P2P) applications using dynamic or random port numbers, the server-client paradigm is still widely prevalent in Web services. Even in P2P applications, the header information of the servers is still important in traffic identification because updating and login operations are processed in 1 or several central servers. Moreover, heavy users of P2P applications could be considered as a kind of servers because of their continuous behavior.

Although HS-based identification can overcome the limitations of the payload-based methods efficiently, it has a significant flaw that the volume of HSs increases rapidly over time as a number of applications emerge, evolve, and vanish.[9,10] The number of possible HS combinations is theoretically 2^{49} (IPv4: 2^{32}; port: 2^{16}; and protocol TCP/UDP: 2), even when we consider only Transmission Control Protocol (TCP)/User Datagram Protocol (UDP) on IPv4 networks. It is impossible to store all the created signatures because of limited memory capacity. For example, the minimum size of an HS is 8 bytes. If all conceivable combinations are extracted as HS, the size of the signature repository exceeds 4.5 petabytes. Moreover, a high volume of stored signatures increases the processing overhead of the identification system. Although hashing mechanism is efficient, an increase in the number of records falling in the same hashing key requires more processing time in the signature matching phase. In addition to the problem of signature storage and computational overhead, accumulating outdated signatures might lower the accuracy of identification results. With applications emerging and vanishing over time, server information should be continuously verified and updated with up-to-date content.[7] Thus, an efficient signature maintenance method that retains only the highly significant signatures in the signature repository is required.

Several works have already used the header information to identify traffic, but their maintenance method was extremely simple and coarse to handle high-speed and large volume Internet traffic.[9–12] We present details of maintenance methods from previous works in Section 2. In this article, we propose an efficient method for HS-based traffic identification and signature maintenance. The proposed identification method consists of 3 parts: signature creation, traffic identification, and signature management. Our approach automatically constructs HSs from the traffic in the training and testing networks where our system is deployed.

The maintenance module in the signature management part only retains the most potentially significant signatures in the signature repository to save memory space and reduce system overhead. For signature maintenance, we define a new metric called the signature weight that reflects its potential ability to identify traffic. The signature weight is periodically calculated and updated to adapt to the changes of the network environment. Our maintenance function calculates a weight for each signature on the basis of a series of maintenance elements: number of counter peer, flow count (FC), and byte count (BC). The maintenance elements used in our maintenance function are defined using statistical information from the identified traffic. Then we delete the signatures deemed less significant on the basis of the calculated weight of a signature at each maintenance interval. We prove the feasibility of the proposed method by developing a prototype system and deploying it in a real operational network. We also define several metrics to evaluate the performance of the maintenance method: number of signatures, completeness, life duration (LD), and execution time. We demonstrate the superiority of our maintenance method by performing a comparison analysis against other existing methods using the above-mentioned evaluation metrics.

The remainder of this article is organized as follows. In Section 2, we review previous methods that have used header information for traffic analysis and analyze their signature maintenance methods. We propose our HS and identification system in Section 3. In Section 4, we propose our HS maintenance method in detail, and we evaluate the proposed method using several evaluation metrics in Section 5. Finally, we conclude our work and propose a future research direction in Section 6.

2 | RELATED WORK

Traffic identification or classification methods have been evolving continuously to adapt to dynamic network environments. Well-known port applications such as Hypertext Transfer Protocol, telnet, File Transfer Protocol, and Simple Mail Transfer Protocol composed most network traffic in the past. As a result, the port-based identification method that uses well-known ports assigned by the Internet Assigned Numbers Authority[13] could identify traffic with a high level of reliability and accuracy. However, the emergence of applications using random or dynamic port numbers to evade firewalls has reduced the accuracy of the port-based identification method to less than 70%.[11,14]

The payload signature–based traffic identification[15–20] offers sufficient completeness and accuracy by checking the existence of a payload signature in a packet payload. However, this method tends to encounter difficulties related to traffic encryption, computational complexity, and invasion of privacy. To overcome some of the issues of payload signature–based identification, a statistical signature method,[21–25] which uses the statistical characteristics of traffic such as distribution of packet size or interval, is proposed. This method can mitigate some of the issues faced by earlier methods because it investigates only the packet header, not the packet payload, to extract statistical values.

However, the method is limited because it is difficult to distinguish between applications using the same communication engine or application-level protocol, as they have similar statistical features. Another approach is the behavior-based method[14,26–28] that examines the network traffic on the basis of the correlation among flows or patterns of traffic generation. However, it is weak to adopt in a real network. Until now, various methods have been proposed to overcome the limitations faced by previous identification methods.

The header-based method is one such example that uses server information only. Several traffic identification methods using header information have been proposed to supplement the accuracy of the port-based analysis method or to verify identification results.[11–13,29] We also propose several identification methods using HS information in our previous works.[8–10] The common assumption among these works is that an Internet application server provides the same application (or service) continuously for a period. Compared to other methods using payload data or statistical features, traffic identification based on header information has several advantages. Some of the advantages are as follows: First, there is no performance degradation caused by packet loss, fragmentation, or sampling that might occur when gathering traffic. Second, there is no performance degradation caused by payload encryption because the method only investigates the header information in a packet. Finally, the method is extremely fast to identify traffic because it compares the header value located in the fixed offset of a packet. Table 1 lists a comparison of existing header-based methods under the perspective of header attributes, analysis purpose, and header maintenance.

Gringoli et al[29] used packet header information to generate ground-truth traffic data. They assumed that all traffic flows with the same "subtuple" pairs (eg, {destination IP, destination port} or {source IP, destination IP}) are generated by the same application. First, they determined the application name of the traffic data using preexisting identification methods such as the payload-based method. Then they extracted subtuples from identified and unidentified traffic using the subtuple information. They repeated this process until no additional network traffic flow was identified. However, this method did not consider any maintenance of the subtuples.

Moore et al[11] proved the inaccuracy of the port-based identification through various experiments. They used the {host, port} pair instead of the port to identify and validate specific application traffic, such as port scanning and audio streaming. However, they used a simple and inferior management scheme that only keeps header information in memory at the point where the host generates corresponding traffic. In other words, the header information of inactive hosts is immediately removed from memory.

Karagiannis et al[13] used behavior patterns and statistical characteristics to analyze P2P application traffic. To do this, they constructed a P2PIP list that consists of {TCP/UDP IP} or {IP, port} pairs and used them to identify traffic. However, this method fails to consider any maintenance processes for the P2PIP list, which leads to an accumulation of entries on the list.

Baldi et al[6] configured a service table by extracting the network coordination (IP address, TCP/UDP, and port) from the result of existing methods (eg, payload-based method). The network coordination was then used for traffic identification. To maintain the service table, they used a simple timeout-based maintenance method. Each element of the service table was deleted if it was not used for a specified period. For example, if the timeout threshold is set at 60 minutes, elements not used for traffic identification for more than 60 minutes are deleted. The timeout-based method is extremely simple and efficient, but it fails to consider servers that provide service occasionally during long period, such as Web mail and antivirus applications.

We define a 3-tuple (IP address, port, and L4 protocol) within a server to construct our HS and also propose a maintenance process that is based on the characteristics of

TABLE 1 Related work using header-based methods

Paper	Format	Purpose	Maintenance Method
Gt: picking up the truth from the ground for Internet traffic[29]	{dst IP, dst port} {src IP, dst IP}	Additional identification	Not mentioned
Toward the accurate identification of network applications[11]	{host, port}	Verification of identification result	Only remaining active host
Transport layer identification of P2P traffic[13]	{TCP/UDP IP Pairs} {IP, port} pairs	Identification	Accumulated
Service-based traffic classification: principles and validation[12]	{IP address, TCP/UDP port}	Additional identification	Inactivity timeout
Internet application traffic classification using fixed IP-port[8]	{IP, protocol, port}	Identification	Inactivity timeout
Signature maintenance for Internet application traffic identification using header signatures[9]	{IP address, port number, transport layer protocol (TCP/UDP)}	Identification	Verify properties (abnormal, timeout, dominant, and popular)
An efficient method to maintain the header signatures for Internet traffic identification[10]	{IP address, port number, L4 protocol (TCP/UDP)}	Identification	Accumulated weight (number of counter peer, flow count, and byte count)

Abbreviations: IP, Internet Protocol; P2P, peer-to-peer; TCP, Transmission Control Protocol; UDP, User Datagram Protocol.

identified network traffic and the history of signature use in our previous works.[8–10] The first article[8] primarily proposes the HS, called fixed IP-port. This method uses a state transition diagram for the extraction phase with several predefined conditions and a timeout threshold. Therefore, it does not need an additional maintenance process. Although the timeout threshold can be adjusted by the system operator according to the network environment, using an inactivity timeout method has limitations caused by overlooking long-term applications, as mentioned in the previous paragraph. A more sophisticated method[9] uses the properties of the identified traffic and the use history of signatures such as abnormal, timeout, dominant, and popular. This method deletes or retains signatures on the basis of a specified set of conditions. However, there are problems in that the process required to obtain signature characteristics is complex and the accumulated value of signature features does not adapt flexibly enough to reflect changes in the network environment. The latest method[10] uses the weight of the signature, which is measured by a function consisting of maintenance elements such as the number of counter peer, FC, and BC. However, the method has limitations in adapting to a real network because it uses accumulated values starting at the extraction. If a signature representing dominant service evolves or vanishes, the wrong signature remains in the repository for a long time. As a result, the outdated signatures cause low accuracy. Thus, the inactivity time is required to calculate the signature weight.

To summarize, previous works have asserted that packet header information can be effectively used in traffic identification. However, these studies mostly focus on the use of the header information, and insufficient attention is given to the maintenance of this header information. In particular, timeout-based maintenance, which deletes unused entries after a period of inactivity, can be ineffective because it tends to delete the header information of applications that generate traffic periodically. This is because this method does not consider the use properties of applications. Although we propose a maintenance method to analyze the characteristics of signatures in our previous work, the maintenance method is too complicated and has difficulties in adapting to network environment changes. As a result, we developed a new maintenance method that only retains the signatures that are most likely to be used in the near future and removes any other signatures to save memory and reduce maintenance time.

3 | HEADER SIGNATURE–BASED TRAFFIC IDENTIFICATION

In this section, we define the HS and explain the structure of our traffic identification system. The HS consists of a 3-tuple (IP address, port number, and L4 protocol) of a server that provides a specific application (service). Further, a weighted value for each HS is calculated at each maintenance interval.

Packet-based analysis is not able to accurately reflect the overall behavior of traffic as the application behavior becomes more complex. To supplement this, a flow unit is proposed. A flow (*f*) is a set of packets having the same 5-tuple (source IP address, source port number, destination IP address, destination port number, and L4 protocol).[4] Thus, we define F as a set of flows as shown in Equation 1 and $F(X = x)$ as a set of flows having the same property x. For example, $F(\mathrm{dst} = \mathrm{dip}, \mathrm{dstPort} = \mathrm{dport})$ is a set of flows having dip and dport, its destination IP address and port number, respectively, as shown in Equation 2:

$$F = \{f_1, f_2, f_3, \ldots, f_n\}, \tag{1}$$

$$F(X = x) = \{f \mid f \in F, f(X) = x,\ X: 5\text{–tuple}\ \}. \tag{2}$$

Although the flow unit is an effective way to analyze massive traffic compared to packet-based techniques, it does not provide sufficient information regarding the behavior of traffic, such as end point connectivity. This is because it consists of a set of packets generated between the end points of 2 specific hosts. In particular, the flow unit requires additional analysis on the relationship between flows in a server-client model by aggregating associated flows. Therefore, we propose new unit for our method:

$$B = \{b \mid b = \{f \mid f \in F(x), x = \{\mathrm{IP}, \mathrm{Port}, \mathrm{Prot}\}\}\}. \tag{3}$$

We define a bunch (b) as the relationship between flows for effective signature creation. A bunch is a set of flows with the same 3-tuple (IP address, port number, and L4 protocol) shown in Equation 3. In other words, all flows within a bunch connect to a specific server port. The bunch consists of 1 server node and a set of flows between counter peers. To determine server-side host, we apply different methods according to L4 protocol (TCP and UDP). In case of TCP, we inspect flag bits of packet. Typically, TCP session initializes on the basis of 3-way handshake that is a process of negotiation between client and server. Synchronize message, synchronize-acknowledgment message, and acknowledgment message are transferred to one another sequentially. Because we can recognize type of the message using flag bits of the packet, we can identify the direction of initial packet. Thus, we can realize which side is server host. In case of UDP, there is no flag bit mechanism unfortunately. Therefore, we assume that both sides of packet are server-side host and make 2 bunches. Although both side of bunches are created, the client-side bunch is finally eliminated in maintenance process because of their relatively weak statistical characteristic.

Figure 1 shows a sample structure of a bunch with flows between 1 server and 2 client hosts. The double arrow line indicates that the flow is generated between a specific server port and several client ports. The line thickness reflects the statistical characteristics of a flow (packet and BC). For example, bold lines correspond to heavy flows and thin lines correspond to light flows. In addition, the end points of the

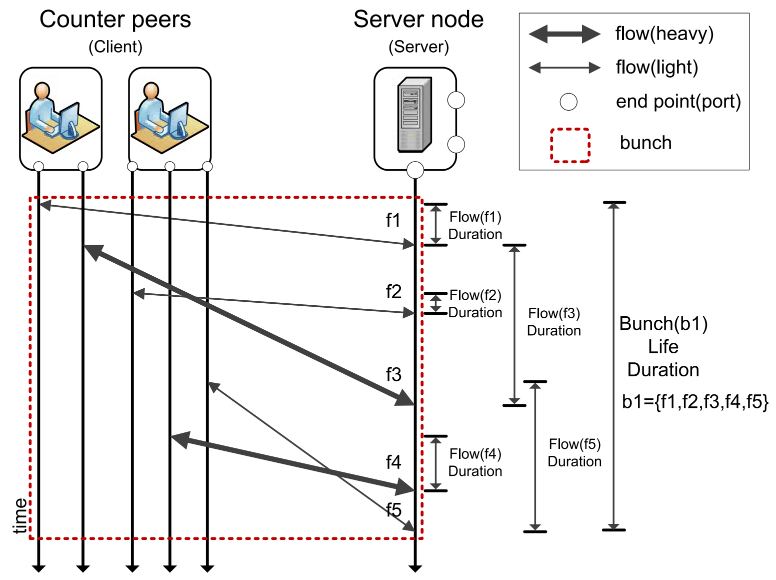

FIGURE 1 Structure of a bunch

arrow lines indicate the start and end times of a flow. All flows in the red dotted box are parts of 1 bunch.

The HS proposed in this article refers to the server node of a bunch, as shown in Equations 4 and 5. The set of HS consists of m HSs. Each HS consists of header information (x), the associated application name (a), and the weight (w) of the HS. The header information is composed of a 3-tuple (IP address, port number, and L4 protocol) of a server-provided application or service. The application name is used to label the traffic identified by the header information (x). The HS weight, the importance level of the signature, is a value measured using the identified traffic for signature maintenance. Thus, the HS is measured by the maintenance function ($M(x)$), which is described in detail in Section 4:

$$\mathrm{HS} = \{hs_1, hs_2, hs_3, \ldots, hs_m\}, \tag{4}$$

$$hs = \{(x, a, w) | x = \{\mathrm{IP}, \mathrm{port}, \mathrm{prot}\}, w = M(x)\}. \tag{5}$$

Table 2 lists the various HS metrics, used as maintenance elements to measure the weight (w) of an HS. An FT denotes the first time a signature is used to identify traffic, and LT denotes the last time. Life duration is the duration between FT and LT. Thus, it refers to the use time of the signature for traffic identification. The FC, PC, and BC mean statistical flow, packet, and BC of identified or extracted traffic flows, respectively, and counter peers count (CPC) corresponds to the number of client hosts in the identified or extracted traffic.

TABLE 2 Header signature metrics

Feature	Description
First time (FT)	First time used to identify traffic, s
Last time (LT)	Last time used to identify traffic, s
Life duration (LD)	Duration of identification (LT-FT)
Flow count (FC)	Flow count of identified (extracted) traffic
Packet count (PC)	Packet count of identified (extracted) traffic
Byte count (BC)	Byte (k) count of identified (extracted) traffic
Counter peers count (CPC)	Number of client hosts in identified (extracted) traffic

Figure 2 shows the architecture of our HS-based identification system. The system consists of 3 parts: signature creation (flow generation, bunch composition, and signature extraction), traffic identification (flow generation, flow identification, and signature feedback), and signature management (signature maintenance and signature naming). In the signature creation part, signatures are extracted from several training networks. If the signature creation part is conducted in certain training network, the created signatures could have network bias in hosts, applications, and network policies of the training network. To avoid this distortion, we conduct the signature creation part in several training networks. The traffic identification part analyzes traffic using the HSs on the target test networks. The reason why we separate training and testing network in Figure 2 is to show the generalization property of HSs generated from a training network, which could be applied to not only the training network but also to any other testing networks without performance degradation. To conduct proposed method in real-world network environment, we should collect multifarious traffic from many different networks for highly generalized HS. However, the traffic collection has some limitations in these days because of privacy issue. Thus, we designed our system architecture having both training and testing networks separately. The signatures generated from training network can be successfully applied to various testing networks even if they are not the training networks. In other words, the proposed method is conducted to maintain HSs in training network, and the signatures as a result of our maintenance

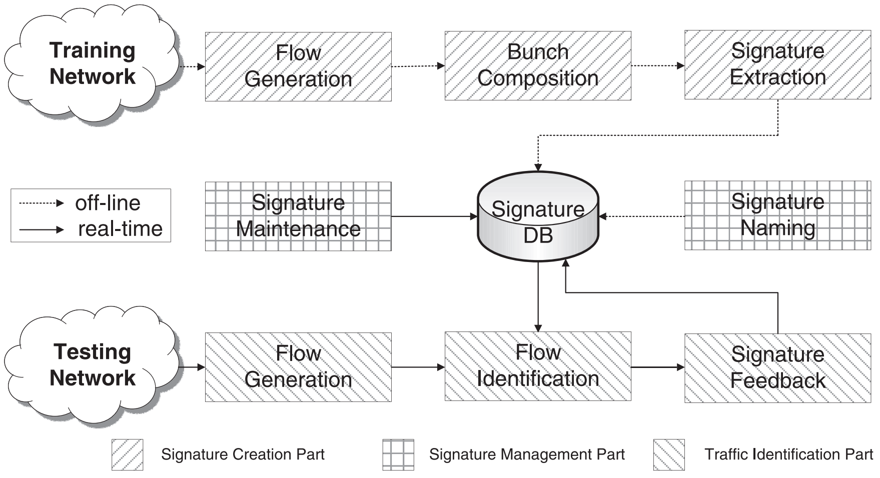

FIGURE 2 Architecture of the proposed header signature–based traffic identification system

method are applied to various testing (target) networks. Finally, the signature management part manages signatures and names the application of signatures.

In the signature creation part, raw packets captured from a training network are reconstructed into flows and bunches, and HSs are extracted. The initial metric values for each extracted signature are set by the statistical values derived from the captured traffic. Subsequently, the signature is added to the signature repository (HS). This creation process can be performed in multiple training networks, either offline or online.

Algorithm 1: Header signature creation

```
Input: $T_{train} = \{t_1, t_2, t_3, \dots, t_{tr}\}$, $HS_{old} = \{hs_1, hs_2, hs_3, \dots, hs_{od}\}$
Output: $F = \{f_1, f_2, f_3, \dots, f_{fl}\}$, $B = \{b_1, b_2, b_3, \dots, b_{bu}\}$,
        $HS_{new} = \{hs_1, hs_2, hs_3, \dots, hs_{nw}\}$
Signature-extraction(T)
1:  for i = 1 to tr            //Flow generation from packets
2:    $F = F \cup$ makeFlow($t_i$)
3:  for j = 1 to fl            //Bunch composition from flows
4:    $B = B \cup$ makeBunch($f_j$)
5:  for k = 1 to bu            //Signature extraction from bunches
6:    $hs_{temp}$ = getServerNode($b_k$)
7:    $hs_{temp}$.updateElement($b_k$)
8:    $hs_{temp}$.setApplication($a$) if possible
9:    If $hs_{temp}$ is new signature
10:     $HS = HS \cup hs_{temp}$
11:   else
12:     updateSignature($hs_{temp}$)
13: return HS
```

Algorithm 1 shows the signature creation process. In signature creation, raw packets (T) consisting of tr packets captured from several training networks are converted into fl flows (flow generation: lines 1 and 2), and these are used to construct bu bunches (bunch composition: lines 3 and 4). For more multifarious signatures, it is acceptable to gather traffic from several networks. Signatures are extracted from the server node (3-tuple) of each bunch (signature extraction: lines 5-12), and the created signatures are stored in a central signature repository. Using the statistical characteristics of the bunch, the metric values of the signatures are updated. The application names (a) of signatures can be named by various identification methods, such as payload or machine learning–based identification, when possible. Signature naming can be performed during the signature creation part as well as during the signature maintenance part. Thus, the application of a signature is named only when it is possible in this part. Created signatures are newly added if they do not exist in the signature repository (HS), whereas existing signatures are updated on the basis of new metric values.

In the signature identification part, it identifies traffic by comparing signatures after reconstructing raw packets captured from a testing network into a specified form (flow unit), and this process is performed in real-time mode. In addition, to manage signatures effectively, the signature maintenance elements are updated on the basis of the statistical values of the identified traffic. These updated values are used when calculating the weight of a signature (w) in the signature maintenance part. The details are provided in Section 4.

Algorithm 2: Header signature-based traffic identification

```
Input: $T_{test} = \{t_1, t_2, t_3, \dots, t_{te}\}$, $HS = \{hs_1, hs_2, hs_3, \dots, hs_{nw}\}$
Output: $F_{identified} = \{f_1, f_2, f_3, \dots, f_{fl}\}$
Signature-identification(T, HS)
1: for k = 1 to te             //Flow generation from packets
2:   $F = F \cup$ makeFlow($t_k$)
3: for i = 1 to fl             //Flow identification
4:   extract {IP,port,prot} = getServer($f_i$)
5:   for j = 1 to nw
6:     if both{IP,port,prot} and $hs_i$({IP,port,prot}) are matched
7:       $F = F \cup f_i$.setApplication($hs_i(a)$)
8:       $hs_j$.updateElement($f_i$)    //Signature feedback
9: return F
```

Algorithm 2 outlines the HS-based traffic identification process. In this process, raw packet (T) consisting of te packets is converted into *fl* flows (flow generation: lines 1 and 2). The header information of each flow is then compared with signatures (flow identification: lines 3-7) stored in the signature repository (HS). Referring to Algorithm 2, the application (*a*) is named to traffic if the traffic and signature have the same header information (*x*). Maintenance elements of existing signatures are also updated (weight feedback: line 8) on the basis of the statistical values of the identified traffic. For readability purposes, Algorithm 2 represents the flow identification process as a brute-force search, which compares the header information against all signatures. In the real implementation, we perform flow identification using a hash data structure to achieve a higher system performance.

In the signature management part of the proposed system, insignificant signatures based on the weight calculated from the maintenance function are deleted, and only the most significant signatures are retained in the signature repository (HS). The applications (*a*) of signatures are also named every maintenance interval if possible. The application (*a*) of a signature is the identity of the traffic when the signature identifies the traffic. For signature maintenance, the signature weight is defined to highlight signatures with high applicability, which is a numerical value representing its applicability. A higher weight value corresponds to a higher applicability for identifying traffic. If the weight of a signature is below the threshold value (0), the signature is removed from the signature repository (HS), as it is considered less likely to be used in the identification.

Algorithm 3: Header signature management

```
Input:  $HS_{old} = \{hs_1, hs_2, hs_3, \dots, hs_{od}\}$
Output: $HS_{new} = \{hs_1, hs_2, hs_3, \dots, hs_{nw}\}$
Signature-maintenance(HS)
1:  for i = 1 to od                //Signature maintenance
2:      hs_i.updateWeight(M())
3:      if hs_i(w) is under the threshold value 0
4:          delete hs_i from signature repository
5:  for j = 1 to nw                //Signature naming
6:      hs_j.setApplication(a) if possible
7:  return HS
```

Algorithm 3 outlines the management process. The HS maintenance updates the weight value of *od* signatures on the basis of the maintenance function (*M*()), which uses maintenance elements (signature maintenance: lines 1-4), and removes any signatures with the weight below the threshold value (0). The application (*a*) of each signature is also named using various methods such as payload, statistical signature, and agent-based methods (signature naming: lines 5 and 6). The maintenance function (*M*()) used to calculate the weight of a signature consists of various maintenance elements. The composition of this function, which is detailed in Section 4, is essential because it determines the identification performance of the entire system. We separated the application naming (*a*) from the signature creation part to create a suitable number of signatures. If we name the application using ground-truth traffic during the signature creation part, the amount of signatures created decreases. Thus, all the header information (*x*) is created from the training network, and then applications are named during maintenance using various methods if possible.

4 | HEADER SIGNATURE MAINTENANCE

The final goal of signature maintenance is to maintain significant signatures in the signature repository that analyzes traffic for a long period. Signature maintenance aims to achieve maximum identification performance (completeness, accuracy, and execution time) with the smallest number of signatures.

Table 3 lists 3 traffic traces captured from our campus network to verify the characteristics of HSs and the feasibility of our maintenance method. First 2 traffic traces were captured during 1 month at the beginning of the semester (KU-Cam-01) and at the end of the semester (KU-Cam-02) from the entire campus network. The last trace was captured during the entire semester (KU-Cam-03) in the selected network.

Figure 3 represents the change in signature counts observed without a maintenance module (accumulation method). That is, it represents the change in signature counts when all generated signatures are stored in a signature repository without deletion. All traffic traces show that the number of signature has steadily grown even in the 3-month trace. This Figure asserts that the number of signatures increases consistently in the absence of a maintenance module.

Because of the limited size of the signature repository, not all of the created signatures can be stored. Thus, we need an efficient maintenance method that maintains signatures identifying large amount of traffic for a significant time and deletes any other signatures from the repository. However, because signature maintenance should be applied in real time, it is difficult to predict whether a signature will be used in the

TABLE 3 Traffic trace

Name	Date	Duration, days	No. of Local IP	No. of Remote IP	Flow, K	Packet, M	Byte, G
KU-Cam-01	03.01.2012	31	10 494	47 040 918	659 694	53 906	47 848
KU-Cam-02	06.01.2012	30	11 437	43 440 289	774 448	59 238	50 704
KU-Cam-03	10.01.2012	92	195	35 924 910	295 071	20 673	16 182

Abbreviation: IP, Internet Protocol.

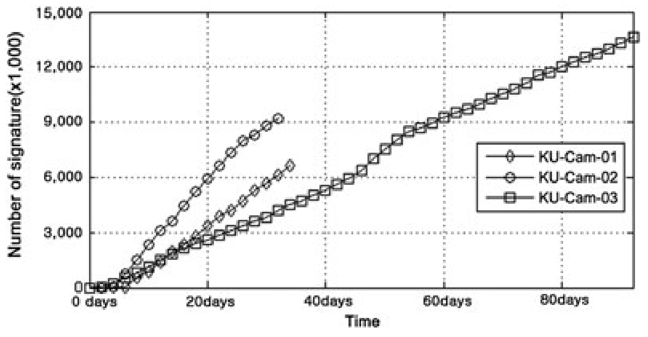

FIGURE 3 Change of signature counts over time

future or not. Thus, in this section, we examine the characteristics correlated with signature LD and define them as maintenance elements. We also propose a maintenance function on the basis of these defined maintenance elements.

To define the maintenance elements, we collected traffic data from our campus network for 1 month (KU-Cam-01) and applied our identification system without a maintenance process. To understand the characteristics of the generated signatures, we renewed the signature metric values listed in Table 1.

Figure 4 shows the distribution of LDs from the generated signatures in the form of a histogram. In the histogram, the *x*-axis refers to the LD of the signatures, and the *y*-axis refers to the number of signatures at each LD in the *x*-axis. For intuitive explanation, we decide intervals of the histogram with time units of 1 minute, 1 hour, 1 day, 1 week, and so on. Each bar represents the number of signatures belonging to each LD. Life duration is specifically defined as the period during which a signature is used to analyze traffic. Signatures with a long LD of more than 2 weeks composed only 6.41% of the total signatures. Conversely, 53.41% of all signatures exhibited an LD of less than 1 minute, which was the minimum time unit measured in the histogram. It is evident that most of the signatures were used in traffic identification only at the point of creation. Accordingly, we defined signatures with an LD of less than 1 minute as flash signatures and further analyzed their characteristics.

Figure 5 shows the correlation between HS LD and FC. The *x*-axis indicates the HS LDs, and the *y*-axis indicates the signature FCs. The numbers in each section represent the number and ratio of signatures that have the corresponding LD and FC. For example, there are 3 795 162 signatures with an LD less than 1 minute and 1 FC. This represents 52.76% of all generated signatures. In addition, 98.78% ($c/(a + c)$) of flash signatures ($a + c$), which have an identification period of less than 1 minute, are associated with an FC of one. On the other hand, only 23.46% ($d/(b + d)$) of nonflash signatures ($b + d$) are associated with an FC of one. Thus, it can be concluded that signatures with a shorter identification period have lower FCs. The correlation between LD and BC is similar to that of LD and FC.

Figure 6 shows the correlation between HS LD and CPC. Counter peers count is defined as the number of unique counter peers in a flow bunch. In Figure 6, the *x*-axis refers to signature LD, and the *y*-axis refers to signature CPC. The numbers in each box represent the total number of signatures at the given intersection points and the percentage of all identified signatures. It is evident from Figure 6 that 99.93% ($c/(a + c)$) of flash signatures ($a + c$) analyzed the traffic from only 1 client (c). On the other hand, only 57.61% ($d/(b + d)$) of nonflash signatures ($b + d$) identified the traffic generated by 1 client (d). After reviewing the data, it is clear that most flash signatures only analyze the traffic from 1 host during their LD.

Table 4 lists the total amount of traffic identified by flash and nonflash signatures, respectively. Flash signatures, which consist of 53.41% of all signatures, identified 1.26% of the total traffic by flow and 0.96% by byte. To summarize, flash signatures have composed most of the created signatures, but in the real analysis, only identified a small percentage of the traffic. In addition, flash signatures only identified a few

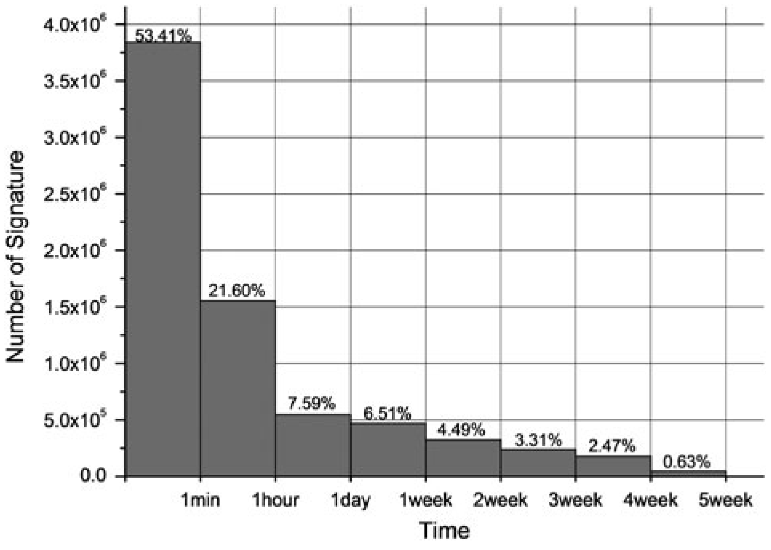

FIGURE 4 Histogram of signature life duration

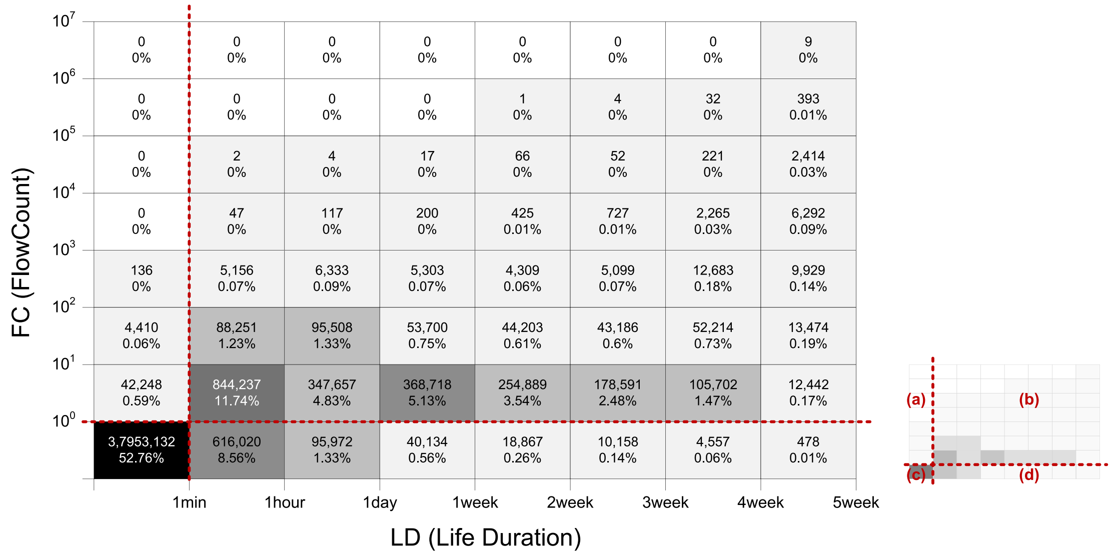

FIGURE 5 Correlation between signature life duration and flow count

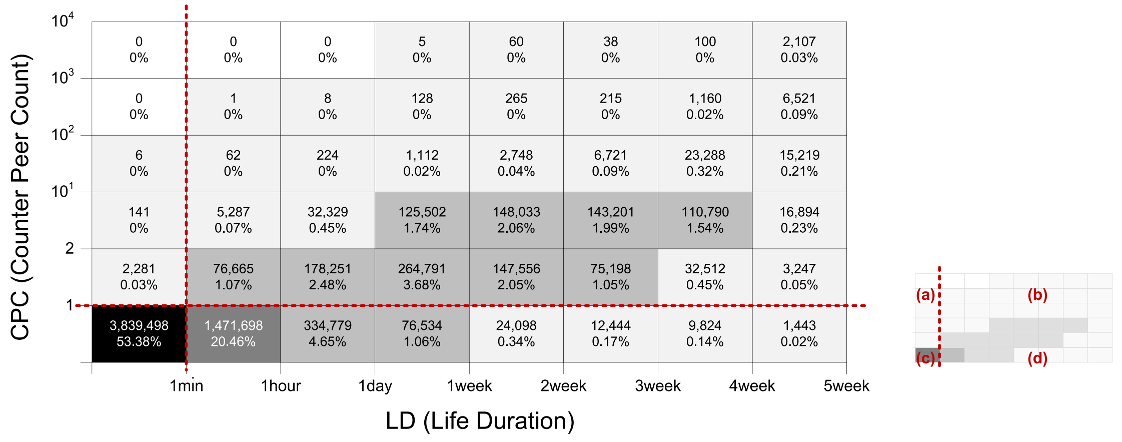

FIGURE 6 Correlation between signature life duration and counter peers count

TABLE 4 Comparison of flash signatures and nonflash signatures

Name	Flash Signatures	Nonflash Signatures	Total
No. of signatures	3 841 926 (53.41%)	3 351 058 (46.58%)	7 192 984 (100%)
Identified flows	3 354 672 (1.26%)	261 761 224 (98.73%)	265 115 896 (100%)
Identified packets, K	332 457 (1.00%)	32 786 180 (98.99%)	33 118 637 (100%)
Identified bytes, M	282 336 (0.96%)	29 093 558 (99.03%)	29 375 895 (100%)

hosts; therefore, it can be concluded that they do not function as signatures that represent specific applications.

We conducted a correlation analysis between signature LDs and several major statistical metrics using Pearson correlation analysis, which is shown in Equation 6:

$$\gamma = \frac{\sum(xy)}{\sqrt{\sum x^2}\sqrt{\sum y^2}}. \tag{6}$$

Table 5 indicates that all the statistical metrics that tested positively are correlated with signature LDs. The statistical metrics of identified traffic, such as FC, PC, BC, and CPC, are correlated with signature LD. In particular, CPC is highly correlated with the signature LD, with a correlation coefficient of 0.684. In order for the maintenance function to be effective at managing signatures, it is necessary that the function reflects the correlation coefficient.

We defined the maintenance elements to be incorporated in our maintenance function on the basis of the HS characteristics, as shown in the experiments described in the previous paragraphs. Specifically, we defined the terms CPC, FC, and BC as maintenance elements to be included in our maintenance function. In our function, hs_t(CPC) is defined as the number of clients from a given traffic identified by an HS

TABLE 5 Correlation coefficients of signature metrics and life duration

Feature	Correlation Coefficient
Flow count (FC)	0.027
Packet count (PC)	0.005
Byte count (BC)	0.007
Counter peers count (CPC)	0.684

(*hs*) at time *t*. Accordingly, hs_t(FC) and hs_t(BC) refer to the number of traffic flows and bytes identified by an HS (*hs*) at time *t*, respectively. The maintenance function, which calculates the weight of each signature, is shown in Equations 7 and 8:

$$M_{\text{proposed}(t)}(hs) = M_t(hs) + M_{\text{proposed}(t-1)}(hs) - (\text{current-time} - hs(\text{LT})), \tag{7}$$

$$M_t(hs) = \alpha \cdot hs_t(\text{CPC}) + \beta \cdot hs_t(\text{FC}) + \gamma \cdot hs_t(\text{BC}). \tag{8}$$

The proposed maintenance function recalculates the weight of each signature (*hs*) at specified time intervals (ϵ). Rather than using cumulative values, each of the maintenance elements (CPC, FC, and BC) is measured at each maintenance interval to appropriately adapt to changes in the network environments. The calculation of the maintenance elements is done in the signature feedback module in Figure 2. Moreover, the calculation of the signature weight based on the maintenance elements and the decision for removing it from the signature repository is conducted in the signature maintenance module in Figure 2.

If a signature does not identify any traffic between maintenance intervals, its weighted value will be decreased. The function subtracts a certain amount (current time − *hs*(LT)) to avoid overloading the signature repository. Because the *hs*(LT) indicates the last time used to identify traffic, the value, current time − *hs*(LT) in Equation 7, will be increased when the signature does not identify any more traffic. Signatures with a positive weighted value are retained in the repository. If a signature does not identify traffic for a span of multiple intervals, its weighted value will be decreased by a larger amount at each subsequent maintenance interval. Signatures with a weighted value under the threshold value (0) are removed from the signature repository by the signature maintenance module. Each maintenance element with the function can control its reflection rate by adjustable constant values (α, β, and γ). This gives network managers the option of altering these values to optimize the function to fit to their network environments.

5 | EXPERIMENT AND RESULTS

We conducted a number of experiments using 1 month of campus network traffic (KU-Cam-01 as listed in Table 2) to evaluate the feasibility of our proposed maintenance function. Because of limitation of physical environment, we could not deploy all methods together in our real operational network. Thus, we collected traffic trace from campus network for 1 month and saved it. The stored traffic trace is divided by a 1-minute file. Consequently, our system processes $60 \times 24 \times 31$ files as input. To analyze the performance of the function, the test was conducted under the assumption that all the extracted signatures could be named at the signature naming module of the proposed identification system illustrated in Figure 2. In addition, the identification part preceded the signature creation part because we targeted the same training and testing networks. The maintenance part was conducted after the creation process. Overall, the identification part, the creation part, and the maintenance part were repeatedly performed at 1-minute intervals. We also included 2 other methods for maintaining signatures that used the HS-based identification to highlight the effectiveness of our proposed maintenance function. The first method included for comparison is the simple accumulation method, which retains all extracted signatures without deletion. As a result, the number of signatures continuously increases, and through this, the accumulation method offers maximum expectation performance of completeness. The second method used for comparison is the timeout-based method.[12] This approach deletes signatures that have not identified traffic for a given period and that exceed a threshold time. Equation 9 shows the weighted measurement formula for signatures under the timeout-based method. The constant value (δ) means the maximum time to remain in the signature repository:

$$M_{\text{timeout}}(hs) = \delta - (\text{current time} - hs(\text{LT})). \tag{9}$$

To perform an objective evaluation of our proposed method, we used the average number of signatures, completeness, the average LD of signatures, and the average execution time as evaluation metrics. In aspect of accuracy evaluation, we checked that there was no decreasing under all methods in our previous works.[8,9] Thus, we only focused on the previously mentioned evaluation metrics:

$$\text{Average number of signatures} = \frac{\sum n(\text{HS})}{t}, \tag{10}$$

$$\text{Completeness} = \frac{\sum \text{Identified traffic}}{\sum \text{Total traffic}}, \tag{11}$$

$$\text{Average LD} = \frac{\sum hs(\text{LD})}{n(\text{HS})}, \tag{12}$$

$$\text{Average execution time} = \frac{\sum \text{Execution time}}{t}. \tag{13}$$

Table 6 lists all evaluation results from the tested maintenance methods: the accumulation, the proposed, and the

TABLE 6 Experimental results

		Completeness			Avg Execution Time, ms	
Maintenance Method	**Avg No. of Signatures, K**	**Flow, %**	**Byte, %**	**Avg LD, min**	**1-core**	**4-core**
Accumulation method	13 190	70.35	89.33	508	1432	1433
Proposed method	911	67.14	89.11	471	46	24
Timeout method	980	66.76	88.21	130	19	18

Abbreviation: LD, life duration.

timeout-based method. We conduct this test several times to find optimal constant values (α, β, γ, and δ). After several tries, we set α, β, and γ as 161 280 ($60 \times 24 \times 7 \times 4 \times 4$) and δ as 1440 (60×24). The constant value applied to the proposed method was larger than that of the timeout-based method but resulted in similar completeness levels. The reason behind the discrepancy in the constant value is that the weight decrement amount performed at each maintenance interval is larger in the proposed method (see Equation 7). The accumulation method showed the best completeness of the 3 methods analyzed, and it resulted in the highest number of signatures. The average LD of the proposed method was longer than the timeout-based method, which means that signatures created in the proposed method were able to identify traffic for a longer period than the timeout-based approach. On the basis of the overall results, the proposed and the timeout-based methods demonstrated similar levels of completeness although they both retained relatively small number of signatures. Furthermore, the average LD for the proposed method was 3 times longer than the timeout-based method. Lastly, the average execution time of proposed method is longer than timeout-based method because of the complexity of the maintenance function, Equation 8, with 1-core central processing unit (CPU). However, we could decrease the execution time closing to that of the timeout-based method by parallel computation with 4-core CPUs. Thus, we believe that the execution time of proposed method is still able to operate in real time as the timeout-based method. This particular case demonstrates significantly the improved performance of the proposed method, and it is analyzed in further detail.

The proposed method retained a lower number of signatures throughout most of the test periods compared to the accumulation and timeout-based method. Figure 7 illustrates the change in the number of signatures stored in the signature repository during the test period by each method. The number of signatures retained by the accumulation method increased dramatically and continuously. Both the proposed and the timeout methods demonstrated fluctuating values in accordance with network traffic but remained well below the levels attained by the accumulation method. On average, our proposed method retained approximately 6% signatures compared to accumulation method. Because the change of signature reflects memory overhead, we can assure that the proposed method is more effective in aspect of memory use. Concerning the number of signature insertion and deletion in the signature repository during 1 minute of maintenance interval, the proposed method gave 200 on average, whereas the timeout-based method gave 1000 on average. The proposed method showed one-fifth less overhead than the timeout-based method. However, this number is negligible comparing to the total number of signatures in repository, which is approximately 950 K on average.

The proposed method showed similar performance (byte completeness) levels compared to the other methods but used approximately 15 times less number of signatures. Figure 8 illustrates the flow and byte completeness of each day of the test period for all 3 methods. Flow unit completeness fluctuated mainly from 60% and 75% throughout the 30-day test periods. Weekday traffic was more complex and higher in volume than weekend traffic because of the nature of the campus network. As a result, the figures from the weekend showed lower levels of completeness. This is because weekend traffic was much lighter and came from fewer hosts. The proposed method demonstrated higher completeness

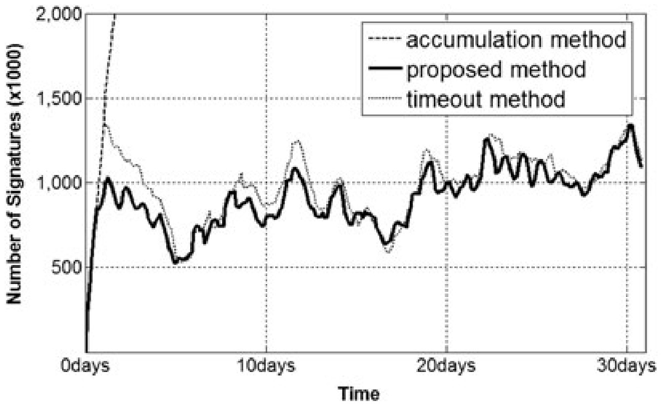

FIGURE 7 Number of signatures over time

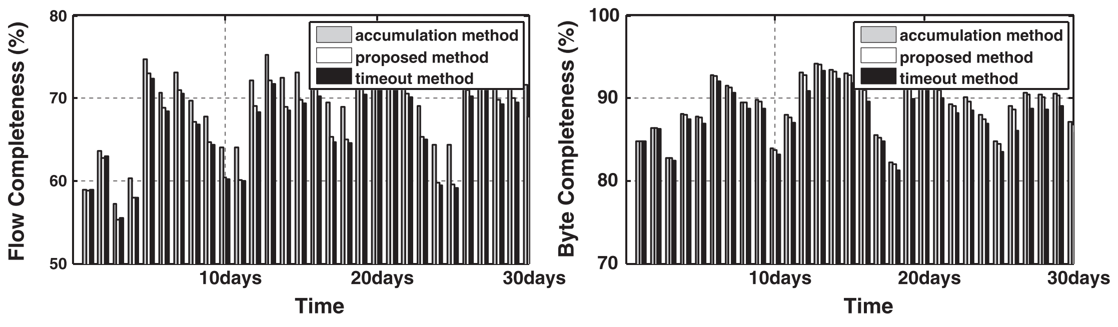

FIGURE 8 Flow and byte completeness over time

than the timeout-based method for most test periods. As reflected in Figure 8, the byte unit values fluctuated between 80% and 95%. The reason why byte completeness was higher than flow completeness is due to the properties of the signatures; most of the heavy flows are identified by the collected signatures. Considering the entire test metrics, the proposed method closely approached the completeness levels of the accumulation method, which is the maximum value we can expect, even outperformed the timeout-based method of completeness throughout the testing period.

Examining this case further, the proposed method was able to identify more traffic by flow and BCs using a smaller number of signatures. Figure 9 shows the cumulative FCs and BCs of the top 1000 signatures from the proposed and timeout-based methods. We selected the top 1000 signatures after sorting the signatures by descending order of identified FCs and BCs at the end of test. The *x*-axis measures the number of signatures, and the *y*-axis measures the cumulative FC and BC values. The signatures managed by the proposed method identified approximately 23 M flows or 2800 GB, whereas the signatures managed by the timeout-based method identified approximately 19 M flows or 600 GB. This shows that the proposed method was better retaining signatures that identified more traffic (especially heavy flows) than the timeout-based method.

Considering the average signature LD during the test periods, the proposed method resulted in higher average durations overall as shown in Figure 10; the average signature LD is 4 times longer than those of the timeout-based method. A longer LD indicates that the signatures stored in the repository were used in traffic identification for a longer period. This means that the proposed method was capable of maintaining and identifying signatures for a long time. In other words, the proposed method has a high possibility to identify intermittent traffic generated by rarely used applications. Figure 10 shows how the average signature LD changed at each maintenance interval during the test periods. In the case of the timeout-based method, LD averages remained low because this approach deleted signatures from the repository periodically.

Test results show that the proposed method was capable of retaining signatures that identify traffic for a long time. This means that the proposed maintenance method was able to identify more traffic in the future. Figure 11 shows the cumulative LDs of the top 10 000 signatures from the proposed and the timeout-based methods. We compiled the

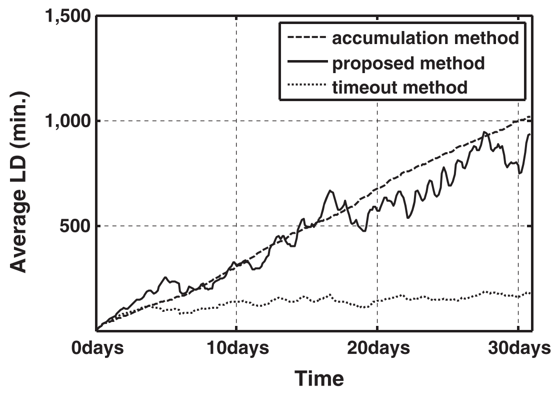

FIGURE 10 Average life duration (LD) over time

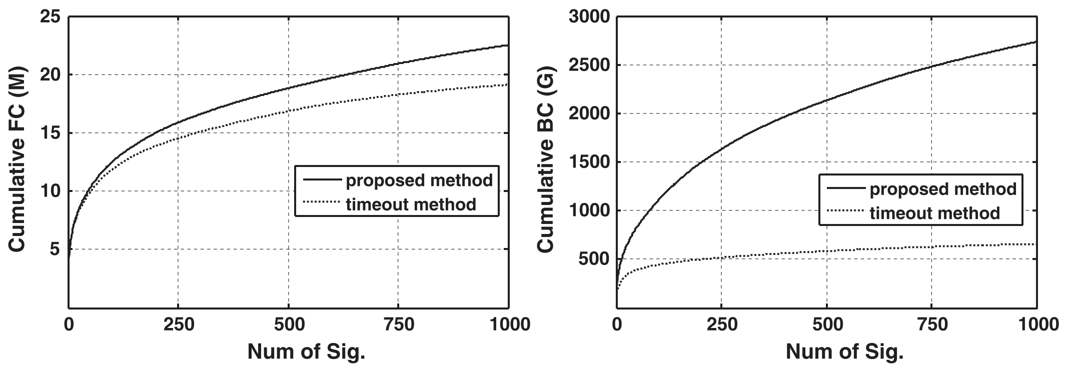

FIGURE 9 Cumulative flow count (FC) and byte count (BC) of the top 1000 signatures

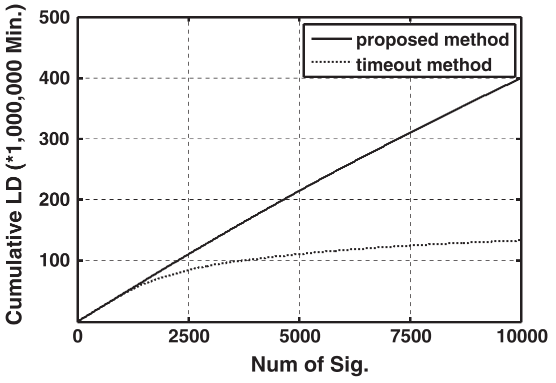

FIGURE 11 Cumulative life duration (LD) of the top 10 000 signatures

top 10 000 signatures after sorting them in descending order at the end of the test periods. The total cumulative LD of signatures managed by the proposed method was 400 000 000 minutes. The average LD of 1 signature was 40 000 minutes (approximately 28 days), indicating that most of the signatures were able to identify traffic during the entire test period. By contrast, the average LD of 1 signature managed by the timeout-based method was only approximately 13 000 minutes (approximately 9 days).

The longer LD also reduces the signature naming overhead, which is out of scope in this article. In case that an HS is added and removed frequently, the signature naming part should be executed to determine the application name of the signature whenever it is added in the signature repository. This could be a significant overhead when we are using payload matching to determine the application name. Also, the short LD makes it delayed to control traffic, such as packet drop, bandwidth shape, and path change, because it takes time to determine the application name.

The proposed method can operate in a real-time environment. Figure 12 shows a comparison of the execution times of the 3 methods during the 30 days of test period. The execution time was measured at the identification and maintenance part of our identification system at every 1-minute maintenance interval. The maintenance part includes all the overhead except the identification overhead, such as the calculation of maintenance equations, the deletions of old signatures, and the insertion of new signatures. Among them, the calculation of the maintenance function occupies most of the maintenance overhead.

For accurate measurement, we checked the total user and system times of target processes. Thus, the execution time can reflect the total CPU use of the methods. The execution time was proportional to the traffic volume and the number of signatures. Accordingly, the execution time for the accumulation method, which had the highest number of signatures, increased continuously. However, the execution time for the proposed and timeout-based methods maintained a steady level throughout the test period.

We measured the execution time of the methods on 2 different CPU platforms: 1-core and 4-core CPUs. For the 4-core CPU platform, we modified the maintenance part of each system with 4 threads executing the maintenance function concurrently. The accumulated method that does not have any maintenance part gave the same execution time, as shown in Figure 12 and Table 7. The timeout-based method that has a very simple equation for maintenance gave very little improvement in total execution time, as shown in Table 7.

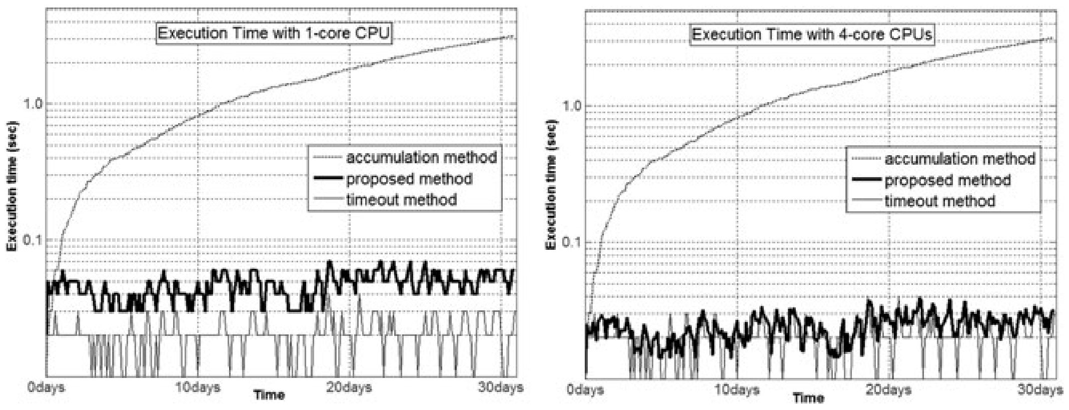

FIGURE 12 Comparison of execution times. CPU indicates central processing unit

TABLE 7 Comparison of execution time

Maintenance Method	Avg No. of Signatures, K	Avg Exec Time with 1-core Ident	Avg Exec Time with 1-core Maint	Avg Exec Time with 1-core Total	Avg Exec Time with 4-core Ident	Avg Exec Time with 4-core Maint	Avg Exec Time with 4-core Total
Accumulation method	13 190	1432	0	1432	1433	0	1433
Proposed method	911	17	29	46	17	7	24
Timeout method	980	18	1	19	18	0	18

Ident, identification; Maint, maintenance.

However, the proposed method gave a high improvement in execution time closing to that of the timeout-based method. We could dramatically reduce it by parallel execution of maintenance functions: Equations 7 and 8. We did not apply parallel execution on the identification part, because it is mainly dependent on the packet capture rate and network speed in real operation network. However, the parallel execution of maintenance part is possible because the maintenance function is periodically executed at the end of each 1 minute.

Although the proposed method required more execution time than the timeout-based method because of the complexity of the maintenance function, this can be reduced by concurrent processing of the maintenance function. Therefore, the proposed method was still able to operate in real time.

The cumulative results from the 1-month test periods showed that the proposed method offered more completeness with fewer signatures when compared to the timeout-based method. In addition, the proposed method further outperformed the timeout-based method by exhibiting a longer average signature LD. The signatures managed by the proposed method were also able to identify more traffic in the flow and byte units for a long period. In addition, the short execution time of the method allowed it to run in real time. On the basis of all the test results, the proposed maintenance method outperformed both the accumulation method and the timeout-based method.

6 | CONCLUSION AND FUTURE WORK

The rapid growth in the number of Internet users worldwide and the popularization of multimedia applications in recent years has emphasized the importance of application-level traffic identification for efficient network management. Various methods for identifying the Internet traffic have been proposed, although it is challenging to apply them to real operational networks because of issues such as signature creation, computational complexity, privacy concerns, and real-time control. To overcome these limitations, we proposed an HS-based traffic identification method. Our method identifies traffic using HSs in 3 main part: signature creation, traffic identification, and signature management. To extract HSs efficiently, we defined a bunch as a set of flows with the same server.

Moreover, we proposed an efficient HS maintenance method that retained only the most significant signatures in a signature repository to promote system optimization. The proposed approach consists of several maintenance elements that were defined after analyzing various statistical metrics of HSs. To prove the feasibility of our proposed method, we used our system in a campus network during the course of 1 month. We defined various evaluation metrics and compared our method with an accumulation and timeout-based method. The evaluation results showed that our proposed method demonstrated similar performance (completeness) to the accumulation method but required 1/25th the amount of signatures. When compared to the timeout-based method, our approach provided higher completeness and used fewer signatures. Furthermore, the signatures managed by our proposed maintenance method were able to accurately identify more traffic by flow and BC for the long term when compared to the timeout-based method. Thus, we were able to prove that our proposed method is more effective than the timeout-based method.

For future work, we plan to extend the empirical experiment to validate the proposed maintenance method in various networks and apply the identification system to additional real training and testing networks. In addition, we plan to analyze a method that can automatically control the constant values (α, β, and γ) in the maintenance function to reflect various target network environments.

REFERENCES

1. Callado A, Kamienski C, Szabo G, et al. A survey on Internet traffic identification. *IEEE Commun Surveys Tuts*. 2009;11(3):37–52. doi: 10.1109/SURV.2009.090304

2. Nguyen TT, Armitage G. A survey of techniques for Internet traffic classification using machine learning. *IEEE Commun Surveys Tuts*. 2008;10(4):56–76. doi: 10.1109/SURV.2008.080406

3. Hyunchul K, KC Claffy, M Fomenkov, D Barman, M Faloutsos, KY Lee. Internet traffic classification demystified: myths, caveats, and the best practices. In: Proceedings of the 2008 ACM CoNEXT Conference, Madrid, Spain, Dec. 2008; 1–12. doi: 10.1145/1544012.1544023

4. Dainotti A, Pescap'e A, Claffy KC. Issues and future directions in traffic classification. *IEEE Netw*. 2012;26(1):35–40. doi: 10.1109/MNET.2012.6135854

5. Dainotti A, de Donato W, Pescapé A. TIE: a community-oriented traffic classification platform. In: *Proceedings of Traffic Monitoring and Analysis (TMA 2009)*. Berlin, Heidelberg;2009:64–74. LNCS 5537. doi: 10.1007/978-3-642-01645-5_8

6. Li W, Canini M, Moore AW, Bolla R. Efficient application identification and the temporal and spatial stability of classification schema. *Computer Netw*. 2009;53:790–809. doi: 10.1016/j.comnet.2008.11.016

7. Aceto G, Dainotti A, de Donato W, Pescap'e A. PortLoad: taking the best of two worlds in traffic classification. In: Proceedings of INFOCOM IEEE Conference on Computer Communications Workshops, San Diego, CA, 2010; 1–5. doi: 10.1109/INFCOMW.2010.5466645

8. Yoon S-H, Park J-W, O Young-Seok, Park J-S, Kim M-S. Internet application traffic classification using fixed IP-port. In: Proceedings of the Asia-Pacific Network Operations and Maintenance Symposium (APNOMS 2009), LNCS 5787, Jeju, Korea, Sep. 2009; 21–30. doi: 10.1007/978-3-642-04492-2_3

9. Yoon S-H, Park J-S, Kim M-S. Signature maintenance for Internet application traffic identification using header signatures. In: Proceedings of the 4th IEEE/IFIP International Workshop of the Maintenance of the Future Internet (ManFI 2012), Hwaii, USA, Apr. 2012; 1151–1158. doi: 10.1109/NOMS.2012.6212042

10. Yoon S-H, Kim M-S. An efficient method to maintain the header signatures for Internet traffic identification. In Network Operations and Management Symposium (APNOMS), 2013 15th Asia-Pacific, 2013; 1–3.

11. Moore A, Papagiannaki K. Toward the accurate identification of network applications. In: Proceedings of Passive and Active Network Measurement (PAM 2005), LNCS 3431, Boston, USA, 2005; 55–68. doi: 10.1007/978-3-540-31966-5_4

12. Baldi M, Baldini A, Cascarano N, Risso F. Service-based traffic classification: principles and validation. In: Proceedings of the IEEE 2009 Sarnoff Symposium, Princeton, NJ, USA, Mar. 2009; 1–6. doi: 10.1109/SARNOF.2009.4850330

13. Karagiannis T, Broido A, Faloutsos M. Transport layer identification of P2P traffic. In: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement, 2004; 121–134. doi: 10.1145/1028788.1028804

14. Karagiannis T, Papagiannaki K, Faloutsos M. Blinc: multilevel traffic classification in the dark. In: Proceedings of the Special Interest Group on Data Communication conference (SIGCOMM 2005), Philadelphia, PA, USA, August 2005; 229–240. doi: 10.1145/1090191.1080119

15. Risso F, Baldi M, Morandi O, Baldini A, Monclus P. Lightweight, payload-based traffic classification: an experimental evaluation. In: Proceedings of International Conference on Communications (ICC 2008), May 2008; 5869–5875. doi: 10.1109/ICC.2008.1097

16. Park J-S, Yoon S-H, Kim M-S. Software architecture for a lightweight payload signature–based traffic classification system. In: Proceedings of Traffic Monitoring and Analysis (TMA 2011), LNCS 6613, Vienna, Austria, Apr. 2011; 136–149. doi: 10.1007/978-3-642-20305-3_12

17. Park B, Won YJ, Hong JW. Toward fine-grained traffic classification. *IEEE Commun Mag*. 2011;49(7):104–111. doi: 10.1109/MCOM.2011.5936162

18. Khalife J, Verdejo J, Hajjar A. Performance of OpenDPI in identifying sampled network traffic. *J Netw*. 2013;8:71–81. doi: 10.4304/jnw.8.1.71-81

19. Yeganeh S, Eftekhar M, Ganjali Y, Keralapura R, Nucci A. CUTE: traffic classification using terms. In: Proceedings of 21st International Conference on Computer Communications and Networks (ICCCN 2012), Munich, 2012; 1–9. doi: 10.1109/ICCCN.2012.6289207

20. Finamore A, Mellia M, Meo M, Rossi D. KISS: stochastic packet inspection classifier for UDP traffic. *IEEE/ACM Trans Netw*. 2010;18(5):1505–1515. doi: 10.1109/TNET.2010.2044046

21. Kumar S, Nandi S, Biswas S. Peer-to-peer network classification using nu-maximal margin spherical structured multiclass support vector machine. In: Proceedings of Data Engineering and Management, LNCS 6411, 2012; 80–84. doi: 10.1007/978-3-642-27872-3_12

22. Yin C, Li S, Li Q. Network traffic classification via hmm under the guidance of syntactic structure. *Computer Netw*. 2012;56(6):1814–1825. doi: 10.1016/j.comnet.2012.01.021

23. Tan J, Chen X, Du M. An Internet traffic identification approach based on GA and PSO-SVM. *J Comput*. 2012;7(1):19–29. doi: 10.4304/jcp.7.1.19-29

24. Dehghani F, Movahhedinia N, Khayyambashi MR, Kianian S. Real-time traffic classification based on statistical payload content features. In: Proceedings of Second International Workshop on Intelligent Systems and Applications (ISA), 2010; 1–4. doi: 10.1109/IWISA.2010.5473467

25. Huang N-F, Jai G-Y, Chao H-C, Tzang Y-J, Chang H-Y. Application traffic classification at the early stage by characterizing application rounds. *Inform Sci*. 2013;232:130–142. doi: 10.1016/j.ins.2012.12.039

26. Hu Y, Chiu D-M, Lui JCS. Profiling and identification of P2P traffic. *Computer Netw*. 2009;53(6):849–863. doi: 10.1016/j.comnet.2008.11.005

27. Cheng WQ, Gong J, Ding W. Identifying BT-like P2P traffic by the discreteness of remote hosts. 32nd Conference on local. *Computer Netw*. 2007;237–238. doi: 10.1109/LCN.2007.69

28. Ullah I, Doyen G, Bonnet G, Gaiiti D. A survey of synthesis of user behaviour measurements in P2P streaming systems. *IEEE Commun Surveys Tuts*. 2011;14:734–749. doi: 10.1109/SURV.2011.082611.00134

29. Gringoli F, Salgarelli L, Dusi M, Cascarano N, Risso F, Claffy K. Gt: picking up the truth from the ground for Internet traffic. *SIGCOMM Comput Commun Rev*. 2009;39(5):12–18. doi: 10.1145/1629607.1629610

How to cite this article: Yoon, S.-H., Park, J.-S., Sija, B. D., Choi, M.-J., and Kim, M.-S. (2016), Header signature maintenance for Internet traffic identification, *Int J Network Mgmt*, doi: 10.1002/nem.1959

AUTHOR BIOGRAPHIES

Sung-Ho Yoon (sungho_yoon@korea.ac.kr, sungho.sky.yoon@lge.com) is a senior research engineer in LG Electronics. He received the B.S., M.S., and PhD degree in computer science from Korea University, Korea, in 2009, 2011, and 2015, respectively. He joined LG Electronics in 2016. His research interests include Internet traffic monitoring and analysis, Internet security, and vehicle security.

Jun-Sang Park (jungsang_park@korea.ac.kr, jungsang.park@lge.com) a senior research engineer in LG Electronics. He received the B.S., M.S., and Ph.D. degree in computer science from Korea University, Korea, in 2008, 2010, and 2015, respectively. He joined LG Electronics in 2015. His research interests include Internet traffic classification and V2X security.

Baraka D. Sija (sijabarakajia25@korea.ac.kr) is an M.S. degree student in the Department of Computer and Information Science, Korea University, Korea. He received his B.S. degree in Information and Communication System from Semyung University, Korea, in 2016. His research interests include Internet traffic monitoring and analysis, service and network management, and Internet security.

Mi-Jung Choi (mjchoi@kangwon.ac.kr) is an associate professor in the Department of Computer Science, Kangwon National University, Korea. She received her B.S. degree in CS from Ewha Womans University in 1998, and M.S. and Ph.D. degrees from the Dept. of CSE at POSTECH in 2000 and 2004, respectively. She was a Postdoctoral fellow at INRIA, France from October 2004 to September 2005 and at School of Computer Science, University of Waterloo, Canada, from November 2005 to October 2006. Her research interests include traffic measurement, M2M network and service management, and mobile abnormality detection and prediction.

Myung-Sup Kim (tmskim@korea.ac.kr) is a professor in the Department of Computer and Information Science, Korea University, Korea. He received his B.S., M.S., and PhD degree in Computer Science and Engineering from POSTECH, Korea, in 1998, 2000, and 2004, respectively. From September 2004 to August 2006 he was a postdoctoral fellow in the Department of Electrical and Computer Engineering, University of Toronto, Canada. He joined Korea University in September 2006. His research interests include Internet traffic monitoring and analysis, service and network management, and Internet security.