Real-Time Measurement of Flows Classified According to their Application for IP Networks

Shan Zhu and Satoru Ohta

***Abstract*—In the management of Internet Protocol networks, the number of flows is an important performance metric because it has useful applications in areas such as port scan detection, denial-of-service detection, and traffic analysis. Real-time counting of flows is particularly important because network operators can take immediate actions against detected network anomalies or performance degradation. This paper presents a method that enables real-time counting of flows classified by application. More useful information for network management can be obtained by counting classified flows. For example, the proposed method is helpful in determining the type of attacks or victim services for attack detection.**

The algorithm for counting classified flows is developed using the timestamp vector algorithm. This paper first explores a naïve method that has as many timestamp vector mechanisms as the application classes. However, this method is disadvantageous because it consumes a very large memory space. To avoid this problem, a new method that considerably decreases memory consumption is proposed. In addition, we also investigate techniques for improving measurement accuracy. The effectiveness of the proposed method is evaluated for real-world network data.

***Index Terms*—application, flows, internet protocol, performance, traffic**

I. Introduction

IN the management of internet protocol (IP) networks, the number of flows is an important performance metric because it has useful applications in areas such as port scan detection, denial-of-service detection, and traffic analysis [1]–[3]. In IP networks, a flow is identified by a flow identifier that is often defined as a five-tuple of source address, destination address, protocol, source port, and destination port in the packet header [2]. Flow counting is defined as a procedure that determines number of different flow identifiers existing in a packet stream.

The number of flows is measurable in a real-time, online manner, while some other flow metrics must be analyzed offline [4]–[6]. This real-time nature of flow counting is particularly significant because network operators can take immediate actions against anomalies found by irregular flow behavior.

Several flow-counting techniques have been proposed [1]–[3]. The number of flows is sometimes counted using a hash function with a bit vector or a Bloom filter [1], [3]. The timestamp vector (TV) algorithm, which also uses a hash function, does not employ a bit vector [2]. Instead, the method uses a TV whose elements show the times of packet arrival. Reference [2] reported that the TV algorithm achieves better accuracy than the method used in [1].

Another important aspect of network measurement is the classification of traffic by the application that generates the traffic [7]–[10]. The advantage of real-time flow counting is enhanced by classifying flows according to the application and then estimating the number of classified flows. By counting classified flows, network operators can determine the most popular services and those that mostly impact network performance. In the case of intrusion and denial-of-service detection applications, which depend on flow counting, the number of classified flows provides valuable information for identifying the types of attacks or victim services. Therefore, it is important to combine flow counting with traffic classification.

This paper presents a method that enables real-time counting of flows classified by application. The number of classified flows can be easily measured by using as many flow counting mechanisms as applications and inputting classified traffic. However, such a naïve method is impractical because of excessive memory space consumption. To avoid this problem, this paper proposes a method that requires smaller memory space. In exchange for memory space savings, the output of the proposed method may include errors induced by the collisions among hash function outputs. However, the errors caused by the collisions are very small. In the proposed method, errors may also occur by the mechanism that identifies flow termination. This paper examines an improved method that judges flow termination more accurately than the existing method. The effectiveness of the proposed method is evaluated through an experiment that uses real-world network data.

The rest of this paper is organized as follows. Section II presents a literature review. Section III identifies the objective of this study. The proposed method is based on the TV algorithm [2]. This basic algorithm is explored in Section IV.

Manuscript received November 27, 2011.

S. Zhu is with the Department of Information Systems Engineering, the Faculty of Engineering, Toyama Prefectural University, 5180 Kurokawa, Imizu-shi, Toyama 939-0398, Japan (e-mail: shanzhu06@hotmail.com).

S. Ohta is with the Department of Information Systems Engineering, the Faculty of Engineering, Toyama Prefectural University, 5180 Kurokawa, Imizu-shi, Toyama 939-0398, Japan (phone: +81-768-56-7500; fax: +81-768-56-6172; e-mail: ohta@pu-toyama.ac.jp).

Then, the proposed method is detailed in Section V. Section VI presents some accuracy improvement techniques. The proposed method is evaluated using real-world network data in Section VII. Finally, we present the conclusions in Section VIII.

II. Related Work

Previous studies have examined methods for counting the number of flows [1]–[3]. Some methods [1], [3] use a hash function and a bitmap. This approach is advantageous because it conserves storage space. The virtual map concept requires an extremely small storage space [1]. This concept enables the bitmap to be stored in high-speed SRAM. However, [2] indicates that the method in [1] is not accurate. As an alternative method, [2] presents the TV algorithm and demonstrates its superiority.

Another important field of network measurement is the classification of traffic by application [7]–[10]. A simple method for classifying traffic is the identification of port numbers found in packets [7]. More sophisticated methods are found in, for example, [8]–[10]. These methods include machine learning-based approaches [8], [9] and signature-based approaches [10]. These approaches classify the traffic more precisely than the port number-based method. However, it should be noted that some of these approaches are developed for offline operation and are too complex to be applied to real-time applications. As an alternative method, a hardware based classification is also available [11].

It is expected that more significant management information will be available if real-time flow counting is combined with traffic classification. However, so far, there have been no studies that agree with this viewpoint.

III. Objective

We develop an algorithm that executes the following processes.

- First, the packets flowing on the monitored link are classified into multiple streams, each of which is generated by a particular application.
- For each classified packet stream, the algorithm counts the number of flows in real-time.

Each flow is identified as a five-tuple of source address, destination address, protocol, source port, and destination port that are stored in the packet header [2]. In other words, the algorithm estimates the number of concurrent sockets that are open for each application at a particular instant.

The classified flows are counted periodically at time t_1, t_2,… Let Δt denote the interval of measurement. That is,

$$\Delta t = t_i - t_{i-1}. \quad (1)$$

The interval Δt is a constant and is set at 1 s in this paper. Let $[t_{i-1}, t_i)$ denote the duration of time t such that $t \in \{t \mid t_{i-1} \le t < t_i\}$. In this paper, the number of flows to be estimated at time t_i refers to the number of existing flows in the duration $[t_{i-1}, t_i)$. Note that some flows may be inactive and will not issue any packets in $[t_{i-1}, t_i)$. Meanwhile, if a flow begins before t_{i-1} and ends after t_i, it certainly exists in $[t_{i-1}, t_i)$. Thus, such a flow is counted even if it does not generate any packets in $[t_{i-1}, t_i)$. This is necessary because the number of inactive, low rate flows may be relevant for network management. In addition, this study assumes that the flows at time t_i include the flows terminated during $[t_{i-1}, t_i)$. Otherwise, the significant information of the short term flows that begin after t_{i-1} and end before t_i will be lost. Fig. 1 illustrates which flows are counted at time t_i.

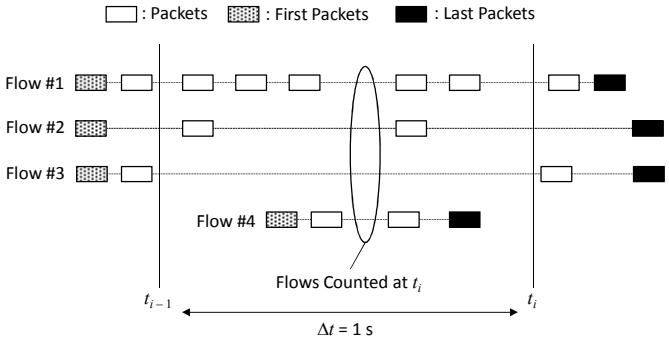

Fig. 1. Flows to be counted at time t_i.

The aim of this study is to derive a method that requires a small memory space and provides an accurate estimation. The developed method works in real-time. Because of this, the method will be applicable to automatic management that requires immediate actions against the anomalies or degradation of network performance.

IV. Basic Algorithm

The proposed method relies on the real-time flow counting method called the TV algorithm [2]. This algorithm is useful to count flows in real-time. The method is based on two mechanisms: a TV (v_0, v_1, …, v_{b-1}) and a hash function that maps the flow identifier (protocol, source/destination address, source/destination port) to integers 0, 1,…, $b-1$. The TV algorithm utilizes the timeout concept to judge the termination of flows. Let T_o denote the timeout period in seconds. Then, the algorithm considers a flow to exist if it generates packets for the last T_o. Using these concepts, the algorithm can be summarized as follows:

1. For each packet arrival, compute the hash function output h from the flow identifier. Then, set v_h to the arrival time.
2. Execute the following periodically at t_1, t_2, … with a fixed interval Δt.
   - Obtain the updated value c, which is the number of vector elements v_i , for the last T_o.
   - Compute the number of flows, denoted as n, by

$$n = b \log_e \frac{b}{b-c}. \quad (2)$$

The TV algorithm is based on the linear counting algorithm, which is comprehensively analyzed in [12]. The derivation of (2) is found in [2], [12].

V. Counting of Classified Flows

This section describes how the TV algorithm is modified to count flows classified by application. The classified flows can be easily counted using a naïve method, described in the next section, which employs as many flow counting mechanisms as the application classes. However, this naïve method is impractical because it consumes a very large storage space. To overcome this difficulty, we present an improved method.

This study employs a simple method based on port numbers shown in packet headers for traffic classification [7]. Complicated classification methods are not appropriate for real-time implementation, and thus, are not discussed. Nevertheless, the port based classification technique is not essential for the proposed method. The method can count flows from a packet stream even if packets are classified by other approaches. Thus, the proposed method may be combined with an approach such as the machine-learning-based classification [8], [9] or the signature-based classification [10], if it performs in real-time. A hardware-based classification [11] will also be usable with the proposed method.

A. Naïve Method

Assume that each packet arriving at the monitored link is classified by application. In addition, assume that there are m applications $a_1, a_2, \ldots, a_m$. Let n_i ($1 \le i \le m$) be the number of flows that belong to application a_i.

The flow numbers $n_1, n_2, \ldots, n_m$ are counted by employing m distinct TVs $\boldsymbol{v_1}, \boldsymbol{v_2}, \ldots, \boldsymbol{v_m}$, each of which is associated with an application. For a packet whose application is classified as a_i ($1 \le i \le m$), the hash output h is computed and then the h-th element of $\boldsymbol{v_i}$, which is denoted as $v_{i,h}$, is updated. By counting c_i, the number of elements updated for the last T_o in $\boldsymbol{v_i}$, and using (2), n_i is represented by

$$n_i = b \log_e \frac{b}{b - c_i}. \tag{3}$$

We will refer to this method as the "naïve method" hereafter. The disadvantage of the naïve method is its excessive consumption of memory space. The TV size b must be very large in order to achieve accurate estimation. Meanwhile, the naïve method requires m times more space to store timestamps compared to the original TV algorithm. This is particularly critical if it is necessary to distinguish among many types of applications.

B. k-Vector Method

The memory required for counting classified flows is considerably decreased by the following method.

The method uses k TVs $\boldsymbol{v_1}, \boldsymbol{v_2}, \ldots, \boldsymbol{v_k}$ ($1 \le k \le m$) as well as additional k vectors $\boldsymbol{s_1}, \boldsymbol{s_2}, \ldots, \boldsymbol{s_k}$. Both vector $\boldsymbol{v_j}$ and additional vector $\boldsymbol{s_j}$ ($1 \le j \le k$) have b elements ($v_{j,1}, v_{j,2}, \ldots, v_{j,b-1}$ and $s_{j,0}$, $s_{j,1}, \ldots, s_{j,b-1}$, respectively). The element $s_{j,h}$ shows which application updates the associated TV $v_{j,h}$. Thus, let us refer to vectors $\boldsymbol{s_1}, \boldsymbol{s_2}, \ldots, \boldsymbol{s_k}$ as "application vectors." Using these vectors, the flow number for each application is computed as follows:

1. For each packet arrival, compute the hash function output h from the flow identifier and identify its application a. If there exists an index j that satisfies $s_{j,h} = a$, then set $v_{j,h}$ to the arrival time. Otherwise, find the TV element $v_{j,h}$ that stores the smallest (i.e., the oldest) value among $v_{1,h}, v_{2,h}, \ldots, v_{k,h}$. Then, set $v_{j,h}$ to the arrival time and set $s_{j,h}$ to a.
2. Periodically at $t_1, t_2, \ldots$, obtain value c_i, which is the number of indices (j, h), such that
   - $v_{j,h}$ is updated for the last T_o
   - $s_{j,h}$ is a_i.

   Then, compute n_i, the number of flows for application a_i, using (3).

Since this method employs k timestamp and application vectors, it will be referred to as the "k-vector method" hereafter. Fig. 2 compares the (a) naïve method with the (b) "k-vector" method.

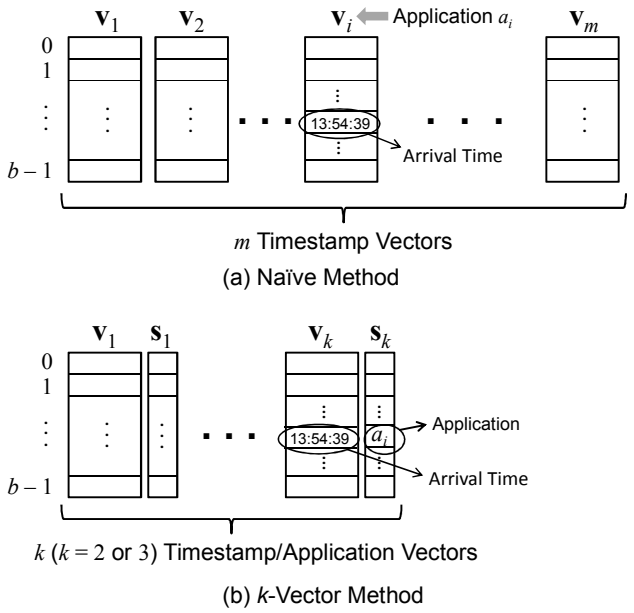

Fig. 2. Data structures for (a) the naïve method and (b) the k-vector method.

If k is set smaller than m, the memory space used by the k-vector method becomes smaller than that used by the naïve method, although the application vectors $\boldsymbol{s_1}, \boldsymbol{s_2}, \ldots, \boldsymbol{s_k}$ for the proposed method are introduced in the k-vector method.

C. Collision Error

If $k < m$, the proposed k-vector method may yield an estimation error. This error is generated when the same hash output is obtained from the flow identifiers of $k + 1$ or more applications. In such a case, an application vector element updated by a flow of an application a_i may be overwritten by that of another application a_j ($i \ne j$, $1 \le i, j \le m$). Then, the element update made by a_i is not recognized by the algorithm. Therefore, c_i and n_i will be underestimated. We refer to this type of error as a "collision error" because it is caused by collisions among hash outputs.

The collision error can be decreased for a larger value of k.

This is shown by the following simple analysis.

Let $P_{\text{coll}}(k)$ denote the probability that a particular hash output value h is obtained from the flow identifiers of $k + 1$ or more applications. Thus, $P_{\text{coll}}(k)$ represents the probability that the collision error occurs for the flows associated with the hash output h. Assume that there are m applications and n flows existing for each application. That is, for simplicity we assume that every application is holding the same number of flows. Then, assuming that the hash function distributes the flow identifier to integers 0, 1, …, $b - 1$ uniformly, the output of the hash value h from one or more flows of an application has the following probability P_h.

$$P_h = 1 - \left(1 - \frac{1}{b}\right)^n \approx 1 - e^{-n/b}. \tag{4}$$

Then, $P_{\text{coll}}(k)$ is represented with P_h as follows.

$$P_{\text{coll}}(k) = \sum_{i=k+1}^{m} \binom{m}{i} P_h^{\,i} (1 - P_h)^{m-i} \tag{5}$$

It is easy to compute $P_{\text{coll}}(k)$ by the relations,

$$P_{\text{coll}}(m) = 0, \tag{6}$$

$$P_{\text{coll}}(k) = P_{\text{coll}}(k+1) + \binom{m}{k+1} P_h^{\,k+1} (1 - P_h)^{m-k-1}. \tag{7}$$

According to (7), $P_{\text{coll}}(k)$ is larger than $P_{\text{coll}}(k + 1)$. Therefore, $P_{\text{coll}}(k)$ decreases for a larger k. This is confirmed by Fig. 3. The figure plots $P_{\text{coll}}(k)$ computed by (4), (6), and (7) against k for the case of b = 10000, 20000, 50000, and 100000. In the figure, n was set at 1000 while m was set at 8. The figure clearly shows that $P_{\text{coll}}(k)$ decreases as k increases. As the figure shows, the probability of the collision error occurrence becomes 10^2 to 10^3 times smaller by increasing k from 1 to 3. This implies that the collision error will be negligibly small for a moderate value of k. This characteristic is further confirmed through an experiment.

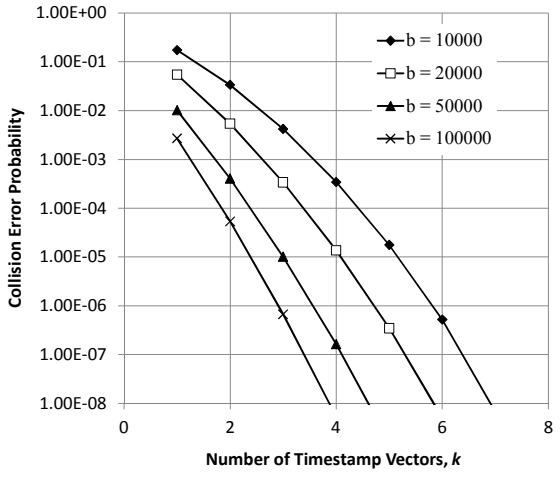

Fig. 3. The probability of a collision error occurrence versus k.

D. Collision Error Evaluation

The analysis described in Section V.C is useful to understand the basic nature of a collision error occurrence. However, it does not present the actual error volume in the output of the k-vector method. To clarify this point, the output of the proposed method was compared with that of the naïve method for real-world network data. For this purpose, the naïve method and the proposed method were implemented using programs running on a Linux OS and written in the C language. These programs identify seven applications by using the port number shown in the packet header. The classified applications (protocols) included world wide web (HTTP), secure web (HTTPS), secure shell (SSH), mail (SMTP), domain name system (DNS), Squid web proxy (Squid), and mail/post office (POP Version3). The employed port numbers are:

80 (world wide web),
443 (secure web),
22 (secure shell),
25 (mail),
53 (domain name system),
3128 (Squid web proxy), and
110 (mail/post office).

For the proposed method and the naïve method, the same modulo-based hash function was used consistently for all applications. The function is defined as follows. Let m_s and m_d denote the source and destination addresses and n_s and n_d denote the source and destination port numbers, respectively. Assume that we are monitoring an IP version 4 packet stream. Then, integers m_s and m_d are four octets long while n_s and n_d are two octets long. In addition, let p be the protocol field value that is one-octet long. Then, for flow identifier $x = (p, m_s, m_d, n_s, n_d)$, the function $h(x)$ is

$$h(x) = (2^{16} p \oplus m_s \oplus m_d \oplus n_s \oplus n_d) \bmod b, \tag{8}$$

where b is set to a prime number.

The programs output the number of classified flows for each application every 1 s. The programs were built using the `pcap` library [13] and can monitor live traffic in real-time and can read `tcpdump`-format files. The effectiveness of the proposed method was evaluated by running these programs. The input for the programs was real-world network data available from the MAWI database of the WIDE project [14]. From the database, a part of the data taken at a sample point F on April 13, 2010 was employed. From the original data file we extracted four 1-h IP version 4 and TCP packet dump files, each of which was created by combining four 15-min files. For these input files, because the programs output the flow numbers every 1 s, 14400 sets of flow numbers were obtained. Then, the root mean squared error was computed between the outputs of the proposed method and those of the naïve method. Fig. 4 shows the result for the case when the application is world wide web. In the figure, the vector size b was set to 80021,120011, and 200003. Similarly, Figs. 5 and 6 show the results for the secure shell and mail applications for the same input data.

Figs. 4–6 show that the error decreases by increasing k as predicted by the analysis. In fact, the error is negligible, even if k is as small as 2. This implies that compared to the naïve method, the proposed method can efficiently save memory space without decreasing accuracy. The figure also shows that

the error decreases by increasing b. This is because the probability of the element update is smaller for a larger value of b.

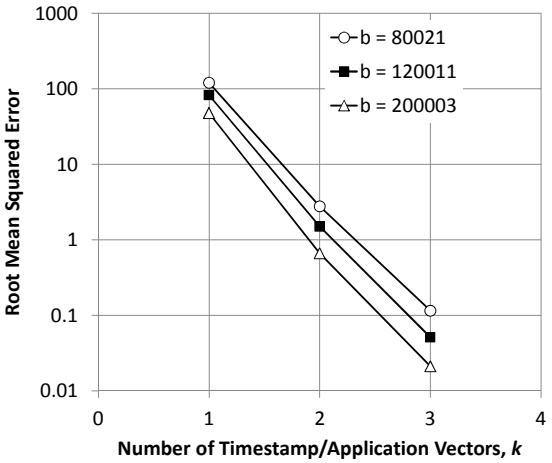

Fig. 4. Root mean squared error of the proposed method for the world wide web (HTTP) application.

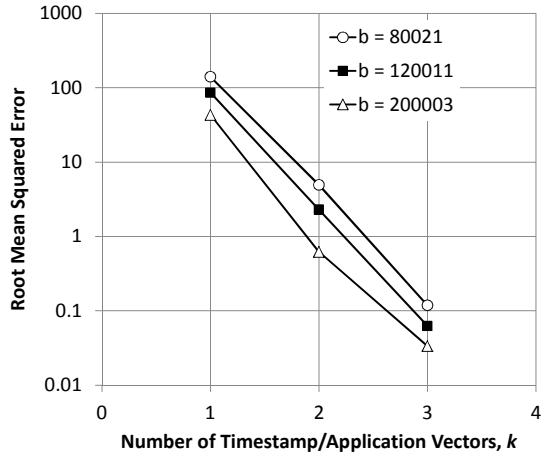

Fig. 5. Root mean squared error of the proposed method for the secure shell (SSH) application.

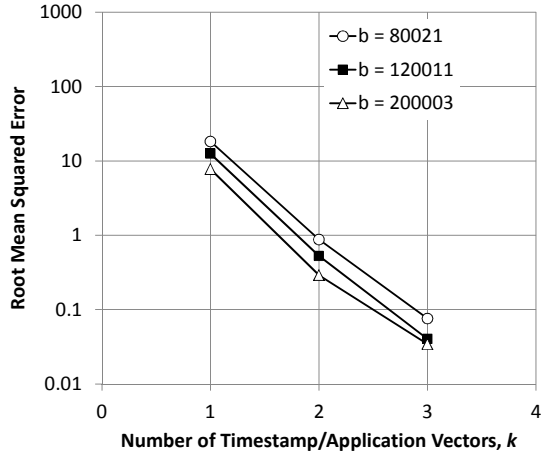

Fig. 6. Root mean squared error of the proposed method for the mail (SMTP) application.

E. Computational Time

The proposed method employs a smaller number of time stamp vectors compared to the naïve method. This means that the number of vector elements that must be checked is smaller for the proposed method than for the naïve method. Therefore, the proposed method is also advantageous from the viewpoint of computational time.

VI. Accuracy Improvement

As shown in the previous section, the collision error of the k-vector method can be made almost negligible by setting k to 2 for a large value of b. However, the output of the algorithm still includes errors resulting from the timeout-based detection of terminated flows. As seen in Section IV, the mechanism considers the flows to be terminated if no packets are issued for the last T_o seconds. This means that the mechanism considers the flows that are terminated for the last T_o seconds to exist. Because of this erroneous termination detection, the algorithm overestimates the number of flows. This overestimation becomes more critical for a large value of T_o. Meanwhile, if the interval between successive packets of a flow is larger than T_o, the detection mechanism considers the flow to be terminated. In this case, the flow is not counted and the total number of flows is underestimated. To avoid this underestimation, the value of T_o must be sufficiently large. This means that the value of T_o must be carefully determined by considering a trade-off between overestimation and underestimation. Unfortunately, it is not easy to select an optimal value of T_o, which avoids both overestimation and underestimation, for any traffic condition.

For TCP flows, the overestimation caused by the timeout-based mechanism is efficiently decreased by considering TCP FIN/RST messages [2], [15]. In a normal operation, a TCP connection is terminated by sending a FIN message. Otherwise, an RST message terminates the TCP connection. Thus, the number of terminated TCP flows can be found by counting the flows that issued FIN/RST messages. Let n_F denote the number of flows that issued FIN/RST messages during the last T_o. Then, the correct number of flows is obtained by subtracting n_F from the flow number estimated by the k-vector (or naïve) method. However, even if the RST/FIN messages are utilized, estimation errors are unavoidable. These errors include the overestimation caused by one-packet flows and the underestimation caused by retransmitted FIN messages.

In addition, some flows are terminated without issuing any FIN or RST messages. For those flows, the termination must be decided with the timeout. Thus, it is important to decide the T_o value adequately. In fact, the optimal value of T_o differs depending on the application because of the difference in packet intervals. This means that it is necessary to set T_o at a different value for each application.

A. One-Packet Flows

Fig. 7 plots the secure shell (SSH) flow number obtained by the proposed k-vector method against time. The figure also

shows the correct flow number, which was obtained by analyzing the output of the `tcptrace` program [16] in the manner described in the Appendix. The data was taken for a part of the packet dump file recorded at the sample point F on April 13, 2010. For the proposed method, we set $k = 2$ and $T_o = 7$ s. The figure shows that the proposed method considerably overestimates the number of flows at the pulse-shaped peak period. By screening the `tcptrace` output, we found many TCP SYN packets without any response during the period. In particular, many flows that issue only one TCP SYN packet exist during the period. We call such a flow a "one-packet" flow. Obviously, the termination of one-packet flows cannot be detected by FIN or RST messages. Therefore, the termination can be only judged by the timeout for these flows. Thus, the algorithm considers a one-packet flow to exist for T_o although it is actually terminated within the transfer time of one packet. This causes overestimation. A similar phenomenon was observed for flows with only two packets.

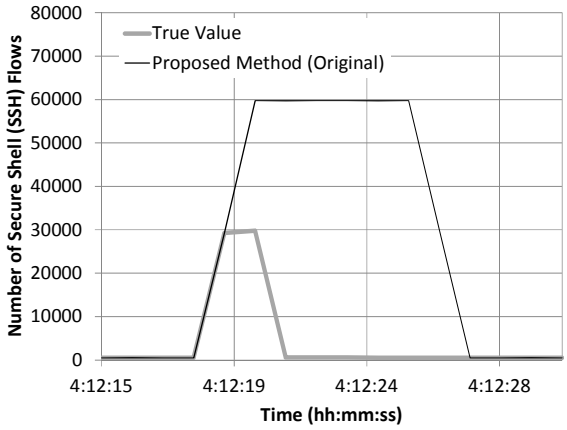

Fig. 7. Overestimation found in the secure shell (SSH) flow numbers of the sample point F data.

In the case shown in Fig. 7, the number of one-packet flows is smaller than 5/s before and after the peak period. The number of one-packet flows suddenly increases to 28,696/s and 29,241/s at 4:12:18 and 4:12:19, respectively. Since the terminations of these flows are not detected for the timeout period, 57,937 (= 28,696 + 29,241) flows are incorrectly counted in the interval. This explains the characteristics shown in Fig. 7 very well.

It is uncertain why one-packet flows increased so rapidly. However, this behavior indicates unusual activities in the peak period. Thus, the detection of such a peak will be very significant for network management.

In real-world networks, one-packet flows are considered common. Such flows will be easily generated if a host tries to make a TCP connection to a nonexistent host or an incorrect address. TCP SYN flood attacks may also generate such flows. Therefore, to further improve accuracy, a mechanism for avoiding overestimation caused by one-packet flows should be added to the algorithm.

The overestimation for one-packet (or two-packets) flows is eliminated by additional k vectors $\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_k$, which indicate the number of timestamp updates. Thus, we define these vectors as "update-count" vectors. At the start of the algorithm, the update-count vectors $\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_k$ are initialized to $\mathbf{0}$. Then, the vector element $u_{i,h}$ is incremented by 1 for each packet arrival, which is associated with a hash value h and the i-th TV. For each periodical calculation of flow number, if the value of element $u_{i,h}$ is 1, the associated flow may be a one-packet flow. If a large value of T_o is applied to this one-packet flow, the flow number will be overestimated, as shown in Fig. 7. Thus, a very short timeout period, denoted by $T_{o,1}$, is applied to $v_{i,h}$ if $u_{i,h} = 1$. Similarly, a short time period $T_{o,2}$ is used if $u_{i,h}$ is 2. For $u_{i,h} > 2$, the timeout period is set to a large value T_o. If the TV element $v_{i,h}$ is not updated for the timeout period, $u_{i,h}$ is reset to 0. With this method, since we can set a sufficiently large timeout value in the case of $u_{i,h} > 2$, it is possible to avoid underestimation for low packet rate flows as well as overestimation for one-packet flows.

As shown in the above procedure, it is unnecessary for $u_{i,h}$ to count values larger than 3. This means that the memory size required for $u_{i,h}$ is as small as 2 bits. Therefore, increase in the memory consumption, by employing the update-count vectors, is very limited.

Fig. 8 shows the effectiveness of using the update-count vectors $\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_k$. In the figure, the characteristic was obtained for the same data as in Fig. 7. The timeout periods $T_{o,1}$, $T_{o,2}$, and T_o were set to 1, 8, and 100 s, respectively. Fig. 8 shows that overestimation is completely eliminated by employing the update-count vectors.

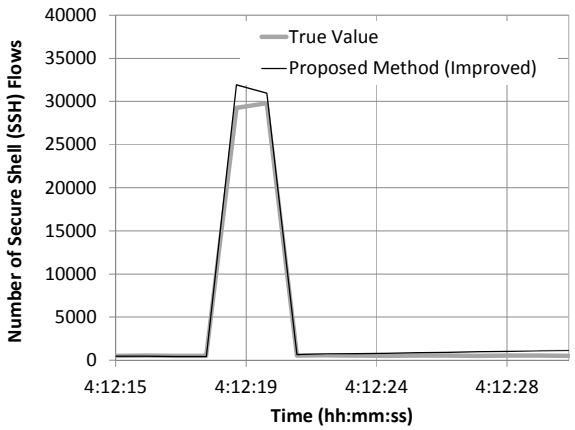

Fig. 8. Elimination of overestimation by update-count vectors.

B. FIN Message Retransmission

When a host closes its TCP connection in a real network, its peer host often does not respond to a FIN message. For this case, the FIN message is retransmitted several times during a considerably long period. Fig. 9 shows an example of the packet sequence of a TCP connection found in the data from the MAWI database. In this example, the FIN message was retransmitted 13 times and then the connection was terminated with an RST message. For this case, suppose that the algorithm

considers the flow to be terminated at the arrival of the first FIN message sent at 0:42:04.712712. Then, the flow is not counted after 0:42:05 even though the connection was not actually terminated. This causes the underestimation of flows.

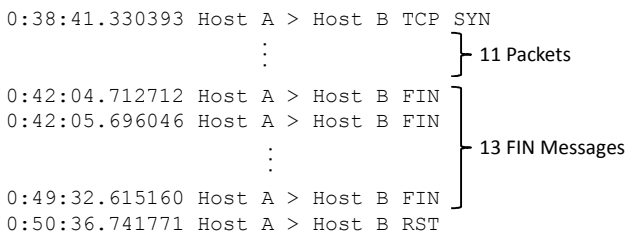

Fig. 9. An example of packet sequence.

Fig. 10 compares the true value of the web proxy (squid) flow number with the value measured by the proposed method with the FIN/RST based termination detection. The figure clearly shows that the number of flows is underestimated. In the figure, 153 flows exist at 0:15:00. Among them, 19 flows transmit four or more FIN messages. Possibly, the algorithm considered these flows to be terminated before 0:15:00 by the first FIN message. Therefore, it is likely that the main cause of the underestimation is the retransmission of FIN messages.

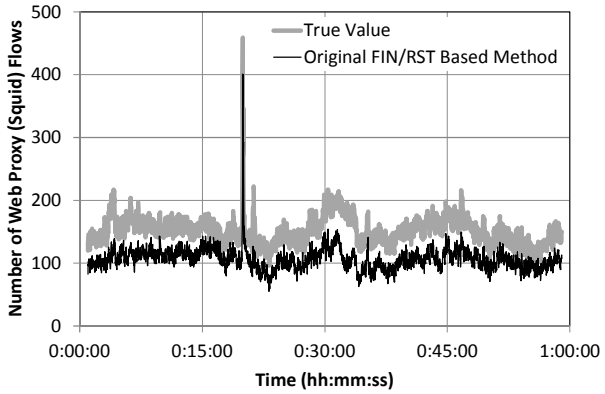

Fig. 10. The underestimation caused by the FIN/RST based termination detection and the retransmission of FIN messages.

The underestimation caused by the FIN message retransmission is improved by adding vectors that count the number of FIN messages. These "FIN-count" vectors $\boldsymbol{f}_1, \boldsymbol{f}_2, \ldots, \boldsymbol{f}_k$ are associated with the timestamp vectors $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k$. The FIN-count vectors act in a similar way as the update-count vectors. Suppose that a FIN packet arrives and the timestamp vector element $v_{i,h}$ is updated. Then, the associated FIN-count vector element $f_{i,h}$ is incremented by one. Each FIN-count vector element is reset to 0 by a timeout. In the flow counting process, if $f_{i,h}$ is smaller than 3, the algorithm considers the flows associated with $v_{i,h}$ to be terminated. Otherwise, the algorithm considers that the FIN message is being retransmitted and the flows are not terminated. That is, for $f_{i,h} > 2$, the termination is detected by the timeout period T_o. If the arrival packet is an RST message, $f_{i,h}$ is set to 1, and thus, the flow is judged to be terminated at the packet arrival time.

Clearly, it follows from the above procedure that each vector element does not need to count a number that is larger than 3. Thus, the memory space increase by adding these vectors is small.

Obviously, this method does not strictly avoid the problem. Nevertheless, experimental results confirm that the method is considerably effective. Fig. 11 shows the characteristic of the termination detection method improved by the FIN-count vectors for the same data as in Fig. 10. The figure clearly shows that the underestimation observed in Fig. 10 is successfully removed by employing the FIN-count vectors.

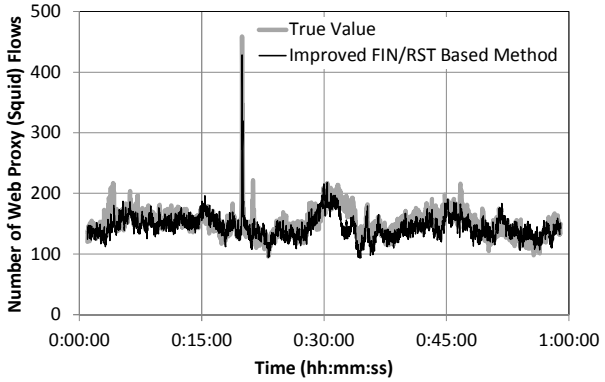

Fig. 11. Characteristic of the FIN/RST based termination detection method improved by employing FIN-count vectors.

C. Timeout Period Setting for Applications

Fig. 12 compares the output of the proposed method with the true number of flows for the Secure Shell (SSH) application. Meanwhile, Fig. 13 shows the output of the proposed method and the true value for the Secure Web (HTTPS) application. In both figures, the timeout period for T_o was set at 60 s. The figures indicate quite different characteristics. That is, the number of flows is overestimated for the SSH case while it is underestimated for the HTTPS case.

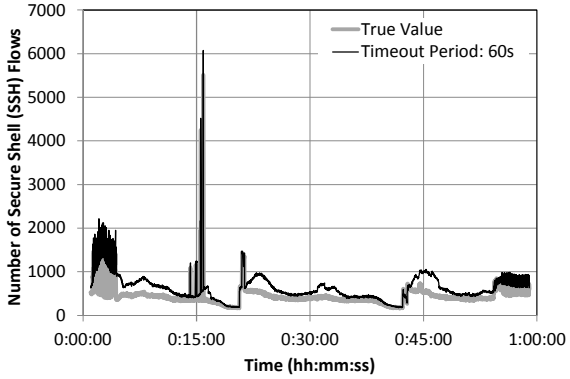

Fig. 12. Secure Shell (SSH) flow number measured by setting T_o at 60 s.

It is impossible to determine the termination of some flows through the FIN or RST messages. Since the termination must be found by timeout for those flows, the decision of the T_o value is important. As stated before, the T_o value must be larger than the packet interval of a flow to avoid underestimation.

However, if the value is too large, overestimation will occur. Meanwhile, the packet interval may not be the same for different applications. Thus, Figs. 12 and 13 imply that the T_o value (60 s) is too large for the SSH flows and is too small for the HTTPS flows.

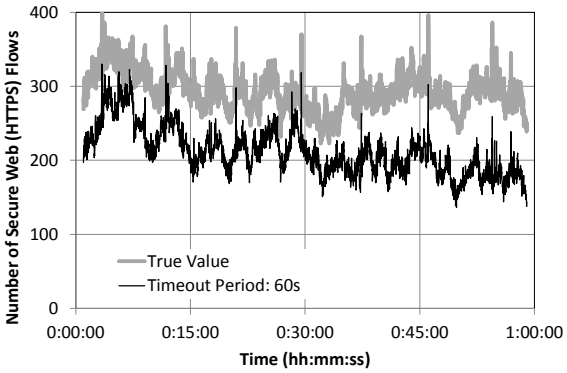

Fig. 13. Secure Web (HTTPS) flow number measured by setting T_o at 60 s.

The above implication is confirmed through Fig. 14. The figure plots the cumulative percentage of the SSH and HTTPS flows against the average packet interval. The average interval was computed from the duration time and the number of packets indicated in the `tcptrace` output. The figure clearly shows that the distribution of the interval greatly differs depending on the application. From the figure, we observe that most SSH flows have short intervals; the average packet interval is smaller than 4 s for 99% of flows. By contrast, a considerable number of HTTPS flows have longer intervals; the average interval value that contains 99% of flows is as large as 32 s. Thus, it is necessary to set T_o at a smaller value for SSH and at a larger value for HTTPS. Actually, the overestimation shown in Fig. 12 and the underestimation shown in Fig. 13 greatly decrease by setting T_o at 15 s for SSH and at 120 s for HTTPS.

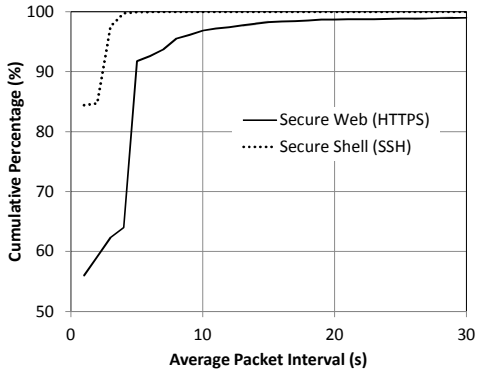

Fig. 14. Cumulative percentage of HTTPS and SSH flows versus average packet interval.

VII. Evaluation

The proposed *k*-vector method with the techniques described in Section VI was implemented using programs written in C language and run on a Linux OS. The method was tested for real-world network data, which is available in the MAWI database.

First, the memory usage of the proposed method was compared with that of the naïve method. The memory usage was measured by executing the "`top`" command while running each program. The vector size b was set at 120011. The measured memory usage is depicted in Fig. 15.

Fig. 15 shows that the memory usage of the proposed method increases as k increases. Nevertheless, even if k = 3, the memory usage of the proposed method is only 42.1 % of that of the naïve method. For k = 2, the memory usage of the proposed method further decreases to 29.5 % of that of the naïve method. Therefore, it is concluded that the proposed method successfully saves memory space.

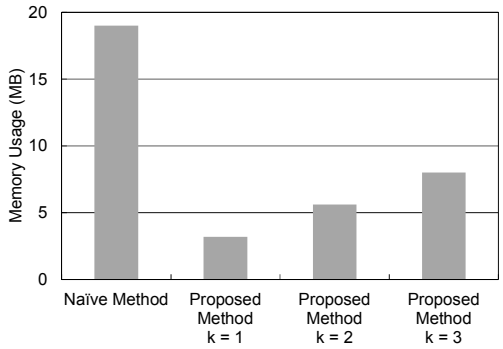

Fig. 15. Memory usage of the naïve method and the proposed method.

The computational time advantage of the proposed method was also assessed through an experiment. The programs were executed for the 1-h packet dump file created from the MAWI data taken at a sample point F on April 13, 2010, and the computational time was measured by the `time` command. Fig. 16 shows the result. As the figure shows, for k = 3, the computational time taken by the proposed method was 34.0 % of that taken by the naïve method. For k = 2, the time taken by the proposed method further decreased to 27.9 % of that taken by the naïve method. This confirms the computational time superiority of the proposed method.

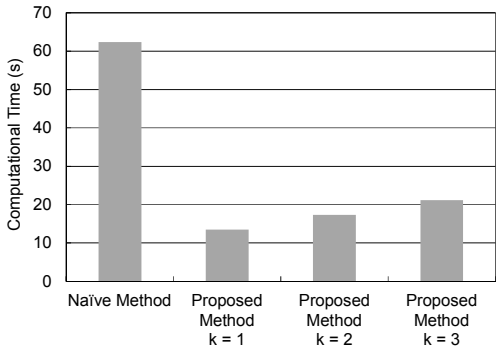

Fig. 16. Computational time of the naïve method and the proposed method.

Next, the accuracy of the proposed *k*-vector method was examined for four 1-h packet dump files, each of which were created by combining four 15-min MAWI database files taken at sample point F on April 13, 2010. As described in Section V,

IP version 4 and TCP packets were extracted from the original data file. The vector size b was set at 120011 and k was set at 2. For one-packet and two-packet flows, the timeout parameters $T_{o,1}$, and $T_{o,2}$ were set at 1 and 8 s, respectively. For other flows, the timeout period T_o must be set at different values for applications as shown in Section VI.C. Thus, T_o was set at 110, 55, 120, 40, 70, 40, and 15 s for DNS, HTTP, HTTPS, POP3, SMTP, SQUID, and SSH, respectively, to minimize the error. The true number of flows was obtained by the method described in the Appendix.

Figs. 17–23 compare the number of flows estimated by the proposed method with the true value obtained from the `tcptrace` output against time for the data from 0:00:00 to 1:00:00. The figures present the characteristics for the DNS, HTTP, HTTPS, POP3, SMTP, Squid, and SSH flows, respectively. As the figures show, the estimation by the proposed method almost coincides with the true value though the characteristics of flows greatly differ among applications. In addition, the time dependency of the flow number is accurately determined when using the proposed method. Therefore, it is expected that anomalies or degradation of the network are successfully detected by the output of the proposed method.

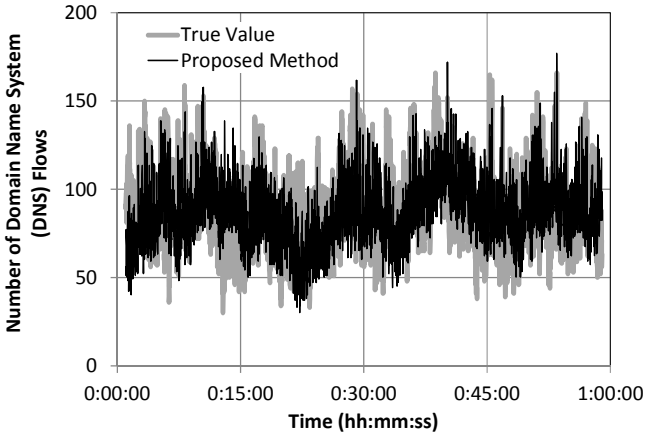

Fig. 17. Domain name system (DNS) flow number estimated by the proposed method for the data from 0:00:00 to 1:00:00.

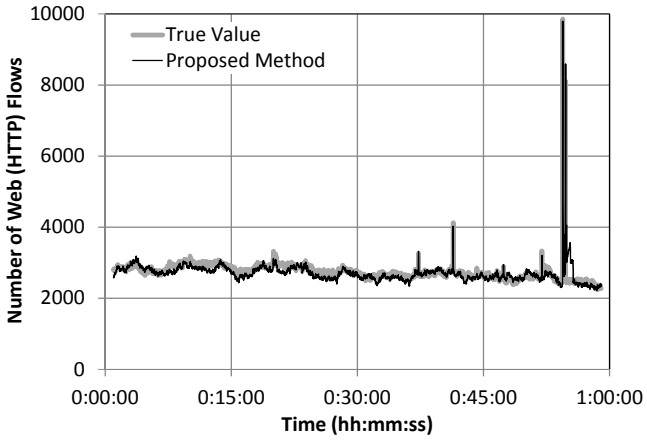

Fig. 18. Web (HTTP) flow number estimated by the proposed method for the data from 0:00:00 to 1:00:00.

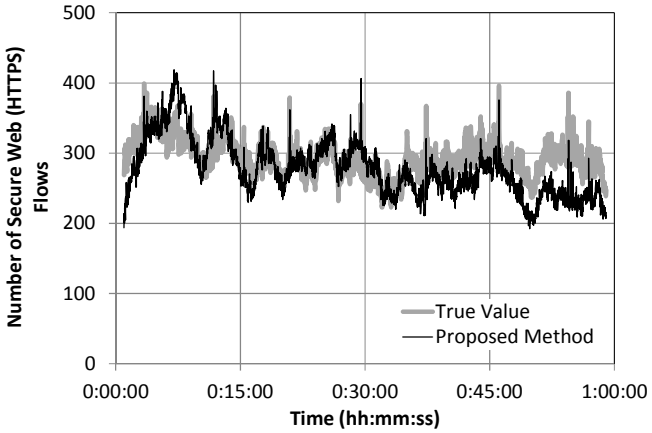

Fig. 19. Secure web (HTTPS) flow number estimated by the proposed method for the data from 0:00:00 to 1:00:00.

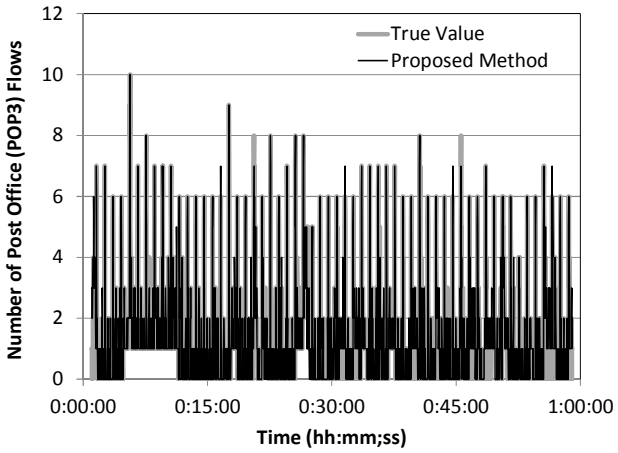

Fig. 20. Mail/post office (POP Ver.3) flow number estimated by the proposed method for the data from 0:00:00 to 1:00:00.

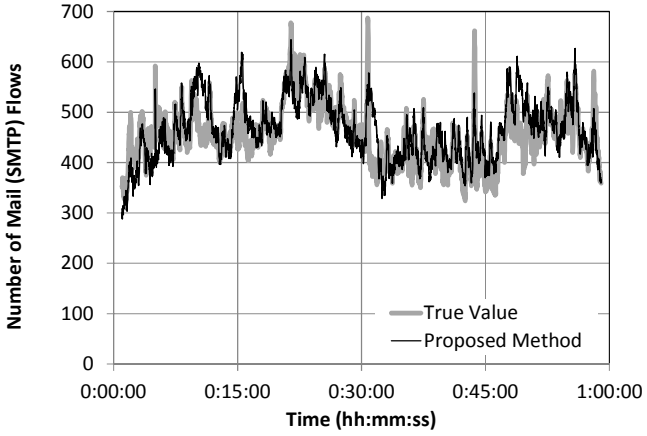

Fig. 21. Mail (SMTP) flow number estimated by the proposed method for the data from 0:00:00 to 1:00:00.

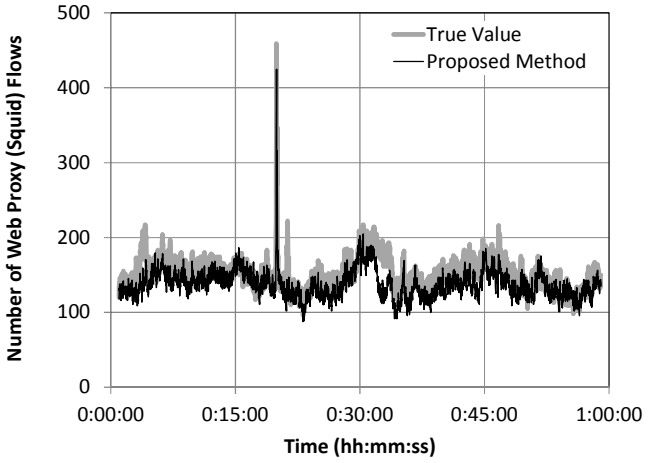

Fig. 22. Web proxy (Squid) flow number estimated by the proposed method for the data from 0:00:00 to 1:00:00.

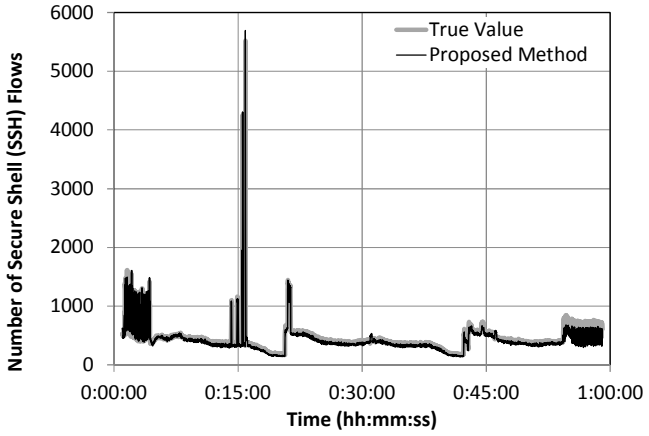

Fig. 23. Secure shell (SSH) flow number estimated by the proposed method for the data from 0:00:00 to 1:00:00.

To confirm the effectiveness under different traffic conditions, the performance of the proposed method was also evaluated for other three 1-h data files, which start at 6:00:00, 12:00:00, and 18:00:00, respectively. Figs. 24–26 show the results for the HTTP flows while Figs. 27–29 show the results for the HTTPS flows. These figures clarify that the estimation by the proposed method is close to the true value in most cases in various time periods, where the traffic load may be very different. This result confirms the reliability of the measurement by the *k*-vector algorithm improved with the techniques described in Section VI. The only exception is shown in Fig. 27, where the flow number is substantially underestimated. This underestimation resulted from the termination detection mechanism of the algorithm. That is, the packet interval of the HTTPS flows became temporarily large for this period and the T_o value was too small. Nevertheless, since the time dependency of the flow number is found exactly in Fig. 27 as well as in the other figures, useful management information is also obtainable through the measurement by the proposed method for this case.

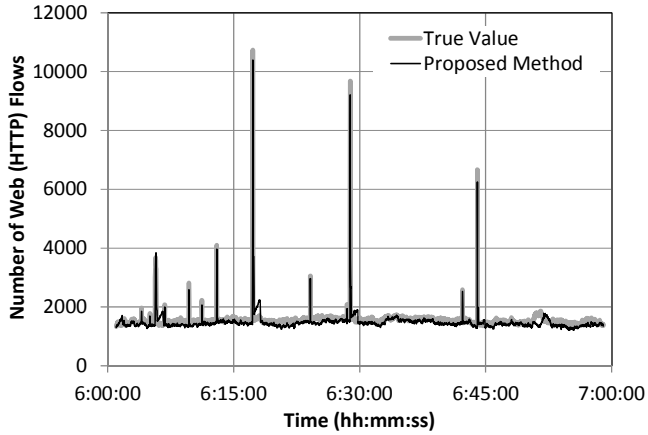

Fig. 24. Web (HTTP) flow number estimated by the proposed method for the data from 6:00:00 to 7:00:00.

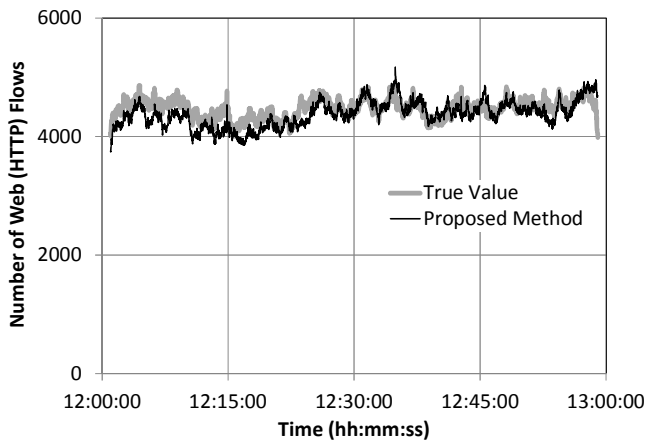

Fig. 25. Web (HTTP) flow number estimated by the proposed method for the data from 12:00:00 to 13:00:00.

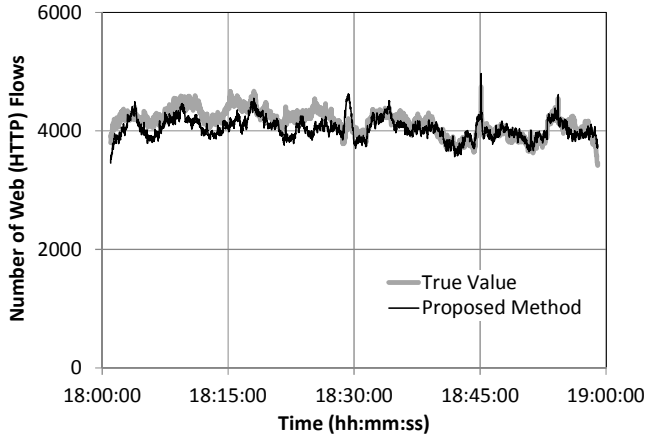

Fig. 26. Web (HTTP) flow number estimated by the proposed method for the data from 18:00:00 to 19:00:00.

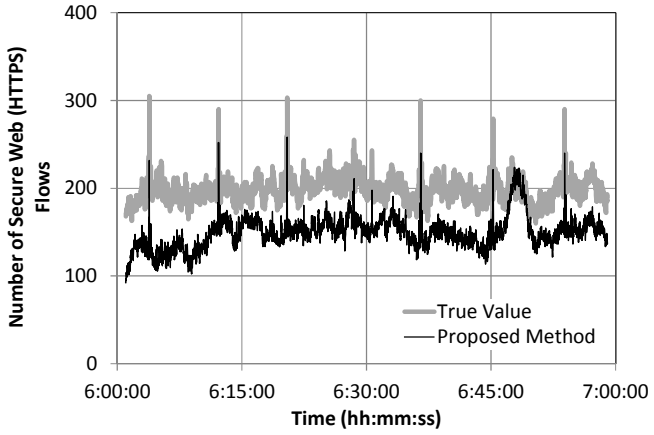

Fig. 27. Secure web (HTTPS) flow number estimated by the proposed method for the data from 6:00:00 to 7:00:00.

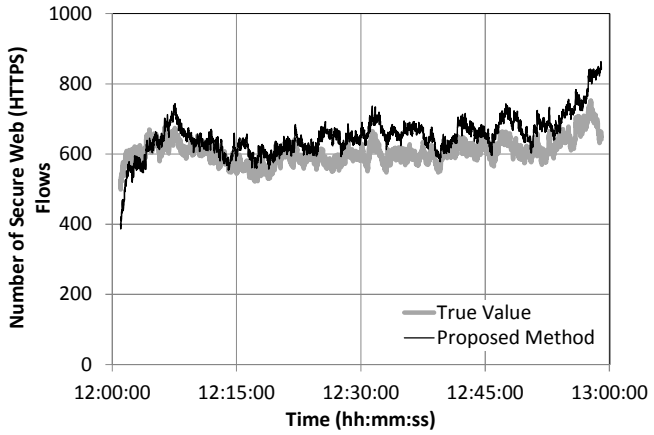

Fig. 28. Secure web (HTTPS) flow number estimated by the proposed method for the data from 12:00:00 to 13:00:00.

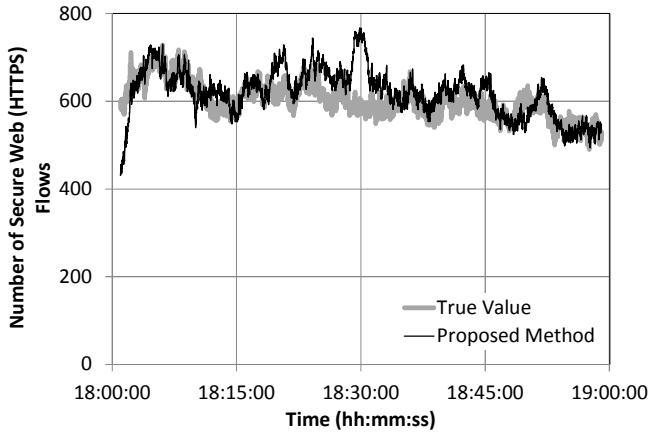

Fig. 29. Secure web (HTTPS) flow number estimated by the proposed method for the data from 18:00:00 to 19:00:00.

Finally, the adequateness of setting k at 2 in the above evaluation is assessed. Fig. 30 compares the estimation obtained by setting k at 1 and 2 by using the naïve method. There is no visible difference between the result for $k = 3$ and that for the naïve method. Thus, the result for $k = 3$ was omitted from the figure. The figure shows that the estimation for $k = 1$ is substantially smaller than that for the naïve method because of the collision error. However, the difference is negligibly small between the proposed method with $k = 2$ and for the naïve method. Thus, it is adequate to set k at 2 for a sufficiently small collision error.

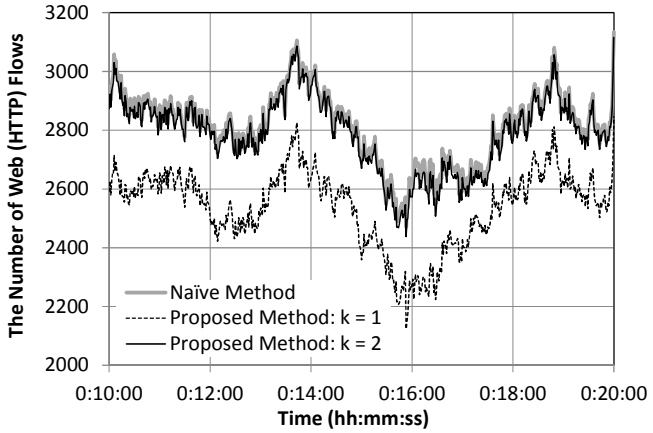

Fig. 30. Outputs of the naïve method and the proposed method with $k = 2$ and $k = 1$.

VIII. Conclusion

This paper first explored a naïve method for counting flows classified by application. Then, the paper proposed a method that employs a smaller number of TVs together with application vectors. The proposed method was compared with the naïve method, which uses as many TVs as applications. As a result, it was confirmed that the proposed method effectively saves memory usage without compromising accuracy. In addition, we also examined techniques that avoid the overestimation for one-packet flows, the underestimation caused by FIN message retransmission and the errors caused by the difference in the optimal timeout periods among applications. Through an experiment, it was confirmed that the proposed method accurately estimates the number of the classified flows.

Appendix

In this study, the true number of flows was estimated as follows. The packet dump file to be tested is first inputted to the `tcptrace` program [16]. The output of the program is stored to a text file in the long format. This output file contains comprehensive information of all flows observed in the dump file. The information includes the arrival times of the first and last packets of a flow. Thus, the duration of a flow is obtained as the period from the first packet to the last packet. Then, the true number of flows at t_i is obtained by counting the flows such that the first packet arrives before t_i and the last packet arrives after t_{i-1}. A Perl script was written to extract the first and last packet time information from the `tcptrace` output file and count the number of flows that should be measured at t_i. It is needless to say that this method cannot be applied to real-time measurement because `tcptrace` is a very slow and memory consuming

program.

REFERENCES

[1] C. Estan, G. Varghese, and M. Fisk, "Bitmap algorithms for counting active flows on high speed links," in *Proc. IMC '03*, Miami Beach, FL, USA, 2003, pp. 153–166.

[2] H. A. Kim and D. R. O'Hallaron, "Counting network flows in real time," in *Proc. GLOBECOM 2003*, San Francisco, 2003, pp. 3888–3893.

[3] K. Keys, D. Moore, and C. Estan, "A robust system for accurate real-time summaries of Internet traffic," in *Proc. SIGMETRICS '05*, Banff, Alberta, Canada, 2005, pp. 85–96.

[4] K. C. Claffy and H. W. Braun, "A parameterizable methodology for Internet traffic flow profiling," *IEEE J. on Selected Areas in Communs.*, vol. SAC-13, Oct. 1995, pp. 1481–1494.

[5] M. S. Kim, Y. J. Won, H. J. Lee, J. W. Hong, and R. Boutaba, "Flow-based characteristic analysis of Internet application traffic," in *Proc. E2EMON*, San Diego, California, USA, 2004, pp. 62–67.

[6] T. Mori, T. Takine, J. Pan, R. Kawahara, M. Uchida, and S. Goto, "Identifying heavy-hitter flows from sampled flow statistics," *IEICE Trans. on Commun.*, vol. E90-B, Nov. 2007, pp. 3061–3072.

[7] S. McCreary and K. Claffy, "Trends in wide area IP traffic patterns - A view from Ames Internet Exchange," in *Proc. ITC Specialist Seminar*, Monterey, CA, USA, 2000.

[8] S. Zander, T. Nguyen, and G. Armitage, "Automated traffic classification and application identification using machine learning," in *Proc. LCN '05*, Sydney, 2005, pp. 250–257.

[9] D. Rossi and S. Valenti, "Fine-grained traffic classification with netflow data," in *Proc. IWCMC '10*, Caen, France, 2010, pp. 479–483.

[10] H. Dahmouni, S. Vaton, and D. Rossé, "A markovian signature-based approach to IP traffic classification," in *Proc. MineNet '07*, San Diego, CA, USA, 2007, pp. 29–34.

[11] *Endace, Capture network packet device*. Available: http://www.endace.com/endace-dag-high-speed-packet-capture-cards.html

[12] K.-Y. Whang, B. T. Vander-Zanden, and H. M. Taylor, "A linear-time probabilistic counting algorithm for database applications," *ACM Transactions on Database Systems*, vol. 15, June 1990, pp. 208–229.

[13] *TCPDUMP/LIBPCAP Repository*. Available: http://www.tcpdump.org/

[14] *Wide: WorkingGroup MAWI*. Available: http://www.wide.ad.jp/project/wg/mawi.html

[15] S. Zhu and S. Ohta, "Fast and accurate flow counting algorithm for the management of IP networks," in *Proc. NOMS 2010*, Osaka, Japan, 2010, pp. 918–921.

[16] *tcptrace - Official Home Page*. Available: http://www.tcptrace.org/

Shan Zhu received the B.E. degree from the Liaoning University of China in 2005 and the M.E degree from the Northeastern University of China in 2008.

She entered Toyama Prefectural University in 2008 and is now a doctor course student.

Ms. Zhu is a student member of the IEICE.

Satoru Ohta received the B.E., M.E., and Dr. Eng. degrees from the Tokyo Institute of Technology, Tokyo, Japan, in 1981, 1983, and 1996, respectively.

In 1983, he joined NTT, where he worked on the research and development of cross-connect systems, broadband ISDN, network management, and telecommunication network planning. Since 2006, he has been a professor in the Department of Information Systems at Toyama Prefectural University, Imizu, Japan. His current research interests are network performance evaluation and power management of network systems.

Dr. Ohta is a member of the IEEE, IEICE and ECTI. He received the Excellent Paper Award in 1991 from IEICE.