Real-Time Flow Counting in IP Networks: Strict Analysis and Design Issues

Shan Zhu and Satoru Ohta

***Abstract*—Real-time flow counting is significant for Internet Protocol (IP)network management because it enables operators to take appropriate action against anomalies or performance degradation. Most flow counting methods proposed in the literature are based on the linear counting algorithm, which was originally developed for database system applications.**

This paper first strictly analyzes the statistical nature of the linear counting algorithm. The correctness of the analysis is confirmed through a computer simulation. The strict analysis is also compared with an approximate analysis reported in a previous study. The result clarifies the conditions where the previous approximate analysis did not provide good accuracy.

The linear counting algorithm is based on a hash function and a vector. To apply this algorithm to flow counting, two design issues arise. One is the method that handles the case of exhausting all vector elements, while the other is the appropriate vector size. The paper presents a simple and effective method for the former. For the latter issue, the upper bound for the flow number is derived as a basis for determining the vector size.

The algorithm is examined for two distinct measurement scenarios: the "active flow scenario" and the "open socket scenario." For each scenario, the estimated accuracy is assessed using real-world network data. As a result, it is shown that the accurate measurement is more difficult for the open socket scenario than for the active flow scenario.

***Index Terms*—IP networks, network management, performance measurement, traffic**

I. Introduction

Flow measurement in Internet Protocol(IP) networks has been studied using various metrics such as flow byte volume, flow packet volume, flow duration, flow timeout, and heavy-hitter flows[1]–[3]. Among these metrics, the number of flows is significant in several useful applications, including port scan detection, denial-of-service attack detection, general measurement in traffic analysis, and the estimation of a TCP connection's throughput[4]–[8].

In IP networks, a flow is identified by a flow identifier, which is defined as a set of fields in the packet header [4], [7]. Flow counting is defined as a procedure that determines how many different flow identifiers exist in a packet stream. The number of flows is measurable in a real-time, online manner, whereas some other flow metrics [1]–[3]must be analyzed offline. Their real-time nature makes the number of flows a particularly useful performance metric.

Flow counting is essential to determine the number of unique values in a large data set. This is efficiently achieved by an algorithm called linear counting, which was comprehensively studied from the viewpoint of database applications [9]. The flow counting methods reported in the literature [4]-[7] are based on this algorithm.

The linear counting algorithm is based on a vector and a hash function. To successfully apply the algorithm to an actual problem, the vector size must be appropriately determined by considering the statistical nature of the algorithm. Several formulas that show the statistical nature of the linear counting algorithm have been presented [9]. However, the formulas are derived using approximation, and thus, are not strictly exact. There have been no studies that assess the accuracy of the approximation sufficiently. Therefore, it is necessary and interesting to evaluate the formulas using a strict analysis. In addition, the algorithm was developed for a database application. Thus, for the flow counting application, the practical design issues of the algorithm must be addressed to satisfy the requirements inherent to IP networks. It is also necessary to evaluate the performance of the algorithm for flow counting in IP networks.

This study investigates the above issues using the flow counting method based on the linear counting algorithm. The first purpose of this study is to strictly analyze the linear counting algorithm and assess the accuracy of the previous approximate formulas. Secondly, the study also focuses on the problems inherent to the flow counting application.

For the first purpose, the paper strictly analyzes the statistical nature of the linear counting algorithm. The analysis is done in a completely different way from the previous study. The accuracy of the analysis is confirmed through a computer simulation. It is also shown that the previous approximate formulas are not always exact depending on conditions such as the problem size. Using strict analysis, it is possible to design the vector size, exactly and independently, for the condition.

To accomplish the second purpose, the paper proposes a new method to deal with the case when the elements of the vector used in the algorithm are exhausted during the counting process. It is confirmed that the proposed method provides an accurate estimation with a simple computational procedure. The paper assesses the upper bound for the number of measured flows. This bound is essential to design the vector size. The flow counting method based on the linear counting algorithm is also tested for real-world network data. Additionally, the paper evaluates the effectiveness of the

Manuscript received January15, 2012.

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)...

accuracy improvement techniques reported in the literature [7], [10]□[12]. By employing these results, it becomes possible to design the algorithm optimally for the flow counting application.

This paper is organized as follows. First, the studies related to this paper are reviewed in Section II. Section III strictly analyzes the statistical nature of the linear counting algorithm. The proposed strict analysis is compared with the approximate formulas of the previous study in Section IV. Section V discusses the design issues when the linear counting algorithm is applied to the flow counting. In Section VI, the flow counting based on the linear counting algorithm is examined for real network data. Finally, Section VII states the conclusion.

II. Related Work

A. Flow-Related Measurement

In IP networks, a flow is identified by a flow identifier, which is defined as a set of packet header fields [4], [7]. This paper defines a flow identifier as a quintuple of source address, destination address, protocol, source port, and destination port, as commonly found in the literature[2], [7]. This definition means that a flow is associated with an open TCP or UDP socket across the monitored link.

Since a flow is a basic unit of communication between application processes, it is important to measure the characteristics of flows for management purposes. Therefore, various flow measurement techniques have been studied [1]–[7]. Reference [1] investigated several flow characteristics including the flow volume and duration. In addition, [1] introduced the concept of an *active* flow; that is, a flow which is active as long as the packets observed are separated in time by less than a specified timeout value. Reference [2] investigates the relationship between the flow characteristics and its applications. In [3], the method of identifying heavy-hitter flows, which issue many packets, is investigated to find the dominant traffic from sampled data.

References [4]–[7] report the flow counting techniques, which estimate the number of flows during a specified time period. While the studies of [1]–[3] present offline approaches, these flow counting techniques are able to provide real-time, online measurements. Because of the real-time nature, the flow counting techniques are important for network operators to take immediate action against anomalies or degradation. In [4], [5], flow counting algorithms based on a bit vector are explored. A similar technique is used in the traffic measurement system described in [6]. Meanwhile, Reference [7] suggests that the method of [4] uses a discrete measurement interval and underestimates the number of flows. To avoid this underestimation, a method called the timestamp vector algorithm is proposed in [7].

It is inadequate to say that the method of [4] underestimates the flow number because the class of counted flows is different between the methods presented in [4] and [7]. The method of [4] exactly estimates the number of *active* flows, which conforms to the definition found in [1]. By contrast, the method of [7] tries to count all existing flows, which include inactive flows in addition to active flows. Because of this difference, it is trivial to estimate that the method of [4] is smaller than the method of [7]. Therefore, we should not easily conclude that the method of [4] is inferior to that of [7]. This paper strictly distinguishes this difference in the flow class.

B. Linear Counting Technique

The flow counting problem is equivalent to counting the number of unique values found in a data set. A practical algorithm for doing this is called the "linear counting algorithm," which is comprehensively analyzed in [9].

The linear counting algorithm is described as follows. The algorithm employs a bit vector of size m. First, all elements of the bit vector are initialized to 0. Each data value is then inputted to a hash function, which maps the input value to an integer from 0 to $m - 1$. The bit vector element whose index is the hash output is turned to 1. Therefore, the value 0 means that the element was untouched during the hash computations. After all data values are processed, the number of unique values is estimated from the number of untouched elements.

Reference [9] derives the following important results by analyzing the linear counting algorithm.

Assume that there are n unique values in the data set. Let U_n be the number of untouched bit vector elements for n unique values. U_n is a random variable. The expected value of U_n, denoted by $E(U_n)$, is

$$E(U_n) \cong me^{-n/m} \text{ for } m \gg 1. \quad (1)$$

Let $\hat{n}$ be the expected number of the unique values. Then, from (1),

$$\hat{n} = -m \log_e \frac{U_n}{m}. \quad (2)$$

We can estimate the number of unique values by $\hat{n}$. Since U_n is a random variable, $\hat{n}$ varies for a fixed value of n and includes an error.

Reference [9] also derived the variance of U_n as follows:

$$\mathrm{Var}(U_n) \cong me^{-n/m}\left\{1-(1+n/m)e^{-n/m}\right\}. \quad (3)$$

Based on (3), the standard error of the ratio $\hat{n}/n$ is estimated as follows:

$$\mathrm{StdError}\left(\frac{\hat{n}}{n}\right) \cong \frac{\sqrt{m}(e^{n/m}-n/m-1)^{1/2}}{n}, \quad (4)$$

where the standard error is defined as the square root of the variance.

The linear count algorithm does not work if all of the vector elements are filled up by 1. If this happens, since $U_n = 0$,the right side of (2) does not have a valid value. Thus, it becomes impossible to estimate n. For the avoidance of this problem, it is essential to considerably decrease the probability that all the elements are filled up by 1. Reference [9] derives this "fill-up" probability by utilizing the fact that the distribution of U_n approaches the Poisson distribution for large values of m and n. That is,

$$P\{U_n = k\} \to \left(\lambda^k/k!\right)e^{-\lambda} \text{ for } m, n \to \infty, \quad (5)$$

where

$$\lambda = me^{-n/m}.$$

Thus, the fill-up probability is:

$$P\{U_n = 0\} \cong e^{-\lambda}. \quad (6)$$

Equations (4) and (6) are particularly important to assess the reliability of the algorithm and to determine the vector

size. It must be noted that these equations are approximations obtained assuming that m and n are large. Thus, the equations may not be sufficiently accurate depending on the values of m and n. Reference [9] compares their approximations with a simulation result to evaluate the accuracy from m = 100 to m = 100,000. However, since the number of trials in their simulation is not large (100 trials), the result is not very reliable. Moreover, the target of their simulation is limited to the estimated value $\hat{n}$ and the standard error. That is, they did not show any results for the fill-up probability. Thus, a more comprehensive study is needed to assess the accuracy of the approximation. In Section IV, the accuracy of these approximate formulas is evaluated using strict analysis.

C. Flow Counting Scenarios

As shown above, flow counting techniques that measure different classes of flows have been reported. This study categorizes these techniques into the "active flow scenario" and the "open socket scenario." Both of these scenarios provide useful information for network management. Each scenario is specified as follows:

(1) Active Flow Scenario

In this scenario, the algorithm counts the number of flow identifiers seen in a specified time period, which starts at time t_1 and ends at time t_2. In other words, only active flows are counted for this scenario. The algorithm does not count the flow that starts before t_1 and stops after t_2, if no packets are between$t_1$andt_2. Therefore, low rate flows may be dropped from the measurement. Though low rate flows may be ignored, this scenario is still useful because it provides the information on active flows, which are influential to the network performance. The flows counted by this scenario are depicted in Fig. 1.

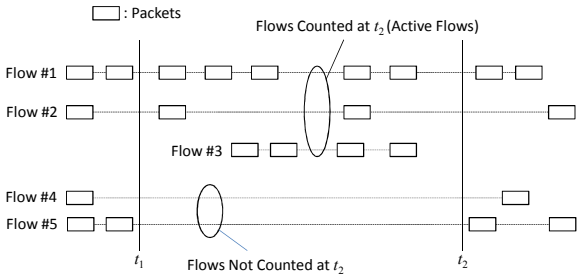

Fig. 1. Flows counted in the active flow scenario.

The linear counting algorithm is applied to this scenario in a straightforward manner. The bit vector elements are first initialized to 0 at t_1. Then, the flow identifier of an arrived packet is inputted to a hash function, and the vector element indexed by the hash output is turned to 1. At t_2, the number of active flows can be obtained by counting U_n and using (2).

The methods of [4], [5] fall into this scenario. The method of [4] is basically identical to the above linear counting algorithm. The method also employs a number of ideas such as the virtual bitmap, the multi resolution bitmap etc, to reduce the memory space.

(2) Open Socket Scenario

In this scenario, the algorithm counts the number of flows that exist on the monitored link at a specified time period. Namely, the number of concurrently open sockets is estimated. Assuming that flows are repeatedly counted at times t_1, t_2,… Then, the open sockets that exist at t_i (i = 2, 3,…) must be counted even if they do not issue any packets during the interval $[t_{i-1}, t_i]$. Fig. 2 illustrates the flows to be counted in this scenario. The open socket scenario is as significant as the active flow scenario because the measurement result will include the information about low rate flows.

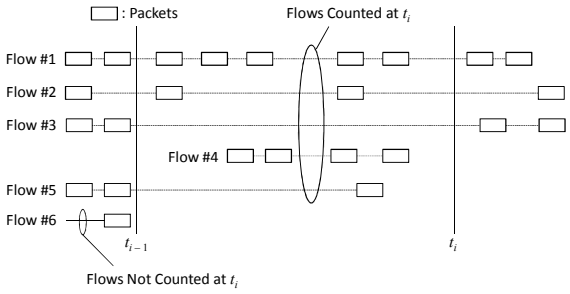

Fig. 2. Flows counted in the open sockets scenario.

To perform this scenario, the algorithm must continuously monitor the packet stream and decide how many flows are generated and not terminated before t_i. Thus, it is important to detect flow termination. Reference [7] presents a method that detects the flow termination through timeouts. This method is based on the linear counting algorithm. However, the method employs a vector of timestamps instead of a bit vector. Because of this, the method is called the timestamp vector (TV) algorithm. When a packet arrives, the method first obtains the hash output from its flow identifier. Then, its arrival time is written to the vector element whose index is the hash output. At measurement time t_i, U_n is obtained as the number of vector elements which are not updated within the timeout period. The number of existing flows is then estimated by (2).

Actually, the termination detection by timeouts is not very accurate. As a method to avoid this inaccuracy, [7] suggests the usage of the TCP FIN field and adapting the timeout period. The effectiveness of employing the TCP FIN was confirmed in [10]. Additional improvement techniques were examined in [11], [12].

III. Strict Analysis

This section strictly analyzes the statistical nature of the linear counting algorithm. The analysis derives the exact probability distribution for the number of bit vector elements turned to 1. This probability distribution is expressed in a recurring form and obtained by iterative computation. Using probability distribution makes computing the standard error and the fill-up probability possible.

Our assumption is that n flows do exist having identifiers, such as$f_1, f_2,\ldots,f_n$. These flow identifiers are mapped to hash values $h_1, h_2,\ldots,h_n$. Amongst$h_1, h_2,\ldots,h_n$, some values may be identical because of a hash collision.

Let p_n be the probability for a set of n flow identifiers that are mapped to a particular hash value vector (h_1, $h_2,\ldots,h_n$).Since a flow identifier is mapped to a particular hash value with probability $p = 1 / m$,

$$p_n = p^n = 1/m^n. \quad (7)$$

Hereafter, set $\{(f_1, h_1), (f_2, h_2), \ldots, (f_n, h_n)\}$ is referred to as a mapping set. Assume that there exist $k(1 \le k \le \min(n,m))$ distinct values H_1, H_2,…,H_k among h_1, h_2,…, h_n. We define $N_{n,k}$ as the number of possible mapping sets between the n flow identifiers and these k distinct hash values. Using $N_{n,k}$, $N_{n,k}\,p_n$ is the probability that the hash values H_1, H_2,…,H_k are generated from n flows f_1, f_2,…, f_n. Let $M_{b,k}$ be the number of sets $\{H_1, H_2, ..., H_k\}$ formed by choosing k distinct numbers from 0, 1,…,$m-1$. Trivially, $M_{b,k}$ is expressed by binomial coefficients,

$$M_{b,k} = \binom{m}{k}. \quad (8)$$

Let $p_{n,k}$ be the probability that k elements of the bit vector are set to 1 by n flows. Using (7) and (8), $p_{n,k}$ is obtained as follows.

$$p_{n,k} = M_{b,k} N_{n,k} p_n = \binom{m}{k} N_{n,k} \frac{1}{m^n}. \quad (9)$$

Equation (9) gives the strict probability that k vector elements are touched by n flows.

Next, let us investigate the characteristics of $N_{n,k}$. First, it is obvious that

$$N_{n,1} = 1. \quad (10)$$

This is because all the n flow identifiers generate the same hash value H_1 for $k = 1$. If, $k = n$,

$$N_{n,n} = n!. \quad (11)$$

In this case, the flow identifier f_1 may generate one of the n hash values H_1, H_2, …,H_n as h_1, and f_2 may then generate one of the $n-1$ values other than h_1. By repeating this observation,(11) is easily derived.

To compute $N_{n,k}$ for $1 < k \le \min(n-1, m)$, let us consider the following two cases:

Case A) The hash value h_n generated by the flow identifier f_n colliding with one or more hash values generated by some of $f_1, f_2, \ldots f_{n-1}$.

Case B) The hash value h_n does not collide with any of the hash values generated by $f_1, f_2, \ldots, f_{n-1}$.

These two cases are illustrated in Fig. 3.There are no other cases in which k distinct hash values are obtained from n flow identifiers. Thus, we can obtain $N_{n,k}$ by summing the number of mapping sets for these cases.

For Case A, k distinct hash values are generated from the $n-1$flow identifiers $f_1, f_2,\ldots,f_{n-1}$. Otherwise, k distinct hash values will not be generated because h_n collides with some of the h_1, h_2, …,h_{n-1} values. The number of mapping sets between $n-1$ flows and k hash values is $N_{n-1,k}$. For each of these mapping sets, h_n may take one of the $k$values, H_1, H_2,…,H_k. Thus, the number of possible mapping sets is $k\,N_{n-1,k}$ for this case.

In Case B, $k-1$ distinct hash values other than h_n are generated from the $n-1$ flow identifiers $f_1, f_2, \ldots, f_{n-1}$because h_n does not collide with any of the h_1, h_2,…,h_{n-1} values. The hash value h_n may take one of the k values, H_1, H_2,…, H_k. For each of these k values, the number of mapping sets between other $n-1$ flow identifiers and the $k-1$ hash values is $N_{n-1,k-1}$. Therefore, the number of possible mapping sets is $k\,N_{n-1,k-1}$ in this case.

From the above consideration, we derive:

$$N_{n,k} = k\,(N_{n-1,k} + N_{n-1,k-1}) \quad \text{for } 1 < k \le \min(n-1, m). \quad (12)$$

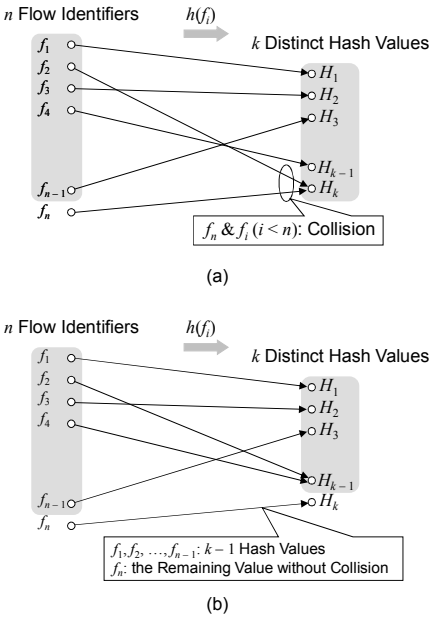

Fig. 3. Possible mappings from n flows to k distinct hash values: (a) Case A and (b) Case B.

The following recurrence formulas are derived using (8), (9),(11), and (12).

$$p_{n,1} = \frac{1}{m} p_{n-1,1} \text{ for } n > 1. \quad (13)$$

$$p_{n,n} = \frac{m-n+1}{m} p_{n-1,n-1} \text{ for } n > 1. \quad (14)$$

$$p_{n,k} = \frac{k}{m} p_{n-1,k} + \frac{m-k+1}{m} p_{n-1,k-1} \quad \text{For } n > 2,\ 1 < k \le \min(n-1, m). \quad (15)$$

Thus, for arbitrary n and k ($1 \le k \le \min(n,m)$), we can calculate the probability $p_{n,k}$ by beginning the computation with $p_{1,1}$ and iteratively applying (13)–(15) while incrementing n. From (9), the initial value of the iteration is

$$p_{1,1} = \binom{m}{1} \frac{1}{m} = 1. \quad (16)$$

If $p_{n,k}$ is known, the standard error and the fill-up probability are immediately obtained. The iterative computation of (13)–(15) is not as fast as the approximate formulas derived in [9]. However, the computational time is less than a few seconds on a PC with a Core2Quad2.83GHz CPU for n< 20000. Thus, this method is considerably practical for a moderate size problem.

To validate the above analysis, a computer simulation was performed. In this simulation, flow identifiers composed of 5-tuples were randomly generated and fed to a hash function on the basis of a prime modulo. The hash function maps a flow identifier to an integer in [0, $m-1$]. The employed hash function is detailed in Appendix A. For the hash output obtained from a flow identifier, the corresponding vector element v_h is set to 1. After executing this procedure for n flow identifiers, the number of untouched vector elements was counted. Repeat in g this procedure yielded the distribution of U_n. The number of repetitions was 10^6 and the

vector size m was a prime number 10007. The distribution of U_n was tested for n = 5000 and n = 30000. That is, the characteristic is assessed for the cases of $n < m$ and $n > m$. The simulation result is compared to the theoretical value obtained from (13)–(16) by setting $k = m - U_n$.

Figs. 4 and 5 show the simulation results. In these figures, the x-axis is U_n, while the y-axis is the frequency of obtaining each U_n value during 10^6 trials. The figure also plots 10^6 $p_{n,k}$ ($k = m - U_n$) as the theoretical value. Fig. 4 shows the characteristic for $n = 5000$, while Fig. 5 shows the characteristic for $n = 30000$. The figures show that the simulation result is very close to the theoretical value for $n < m$ as well as for $n > m$. This confirms the correctness of the proposed analysis.

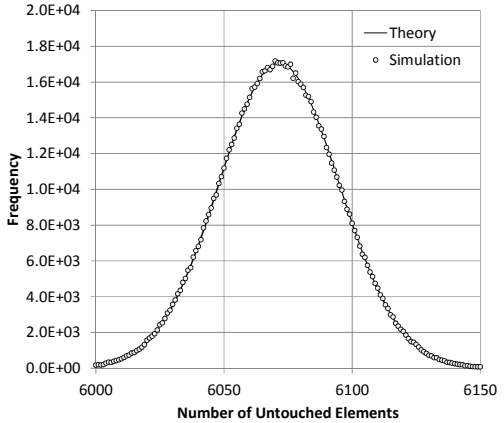

Fig. 4. Distribution of U_n for n = 5000.

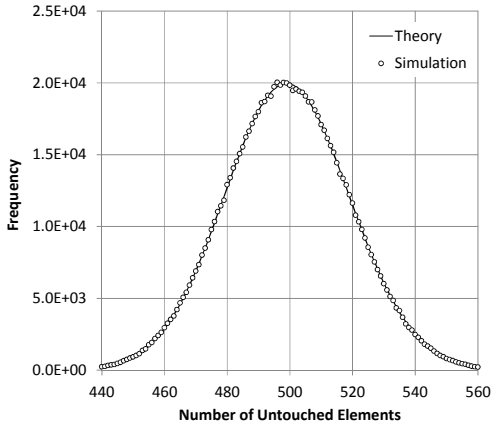

Fig. 5. Distribution of U_n for n = 30000.

IV. Comparison between Strict Analysis and Approximation

This section compares the proposed strict analysis with the approximate formulas derived in [9]. The comparison is performed for the standard error of $\hat{n}/n$ as well as the fill-up probability. For the method of [9], the values are obtained by (4) and (6). For the strict analysis, the standard error of $\hat{n}/n$ is computed by:

$$\mathrm{StdError}\left(\frac{\hat{n}}{n}\right) = \sqrt{\sum_{k=1}^{\min(m-1,\,n)} \left(\frac{\hat{n}_k}{n} - 1\right)^2 p_{n,k}}\ , \qquad (17)$$

where

$$\hat{n}_k = -m \log_e \frac{m-k}{m}\ .$$

The probability $p_{n,k}$ is calculated by (13)–(16). The fill-up probability by strict analysis is:

$$P\{U_n = 0\} = p_{n,m}\ . \qquad (18)$$

In addition, a computer simulation was performed on the analysis. In the simulation, n flow identifiers were randomly generated and the linear counting was executed in each trial. For standard error evaluation, the squared error of $\hat{n}/n$ was computed from the estimated value $\hat{n}$. If the vector was filled up, the data of the trial was not used. The trial was repeated 10^6 times and then the standard error was obtained from the average of the squared errors. For the fill-up probability, the simulation procedure is similar. In this evaluation, the number of vector fill-up events was summed up for 10^6 trials. Thus, the fill-up probability is estimated by dividing the total number of fill-up events by 10^6.

Figs. 6 and 7 show the standard error for a small vector size (m = 101) and a moderate vector size (m = 10007). In the figures, the x-axis is n, while the y-axis is the standard error of $\hat{n}/n$. In Fig. 6, the proposed strict analysis agrees well with the simulation result. This supports the accuracy of the proposed analysis. The given figure shows that the formula of [9] considerably underestimates the error for a larger value of n. This is predictable because the formula is valid only for large values of m. That is, the method is not very accurate if m is as small as 101.In Fig. 7, the method of [9], the proposed analysis and the simulation result show almost the same standard error values. This clearly shows that the formula developed in [9] provides a very good approximation if m is as large as 10007.

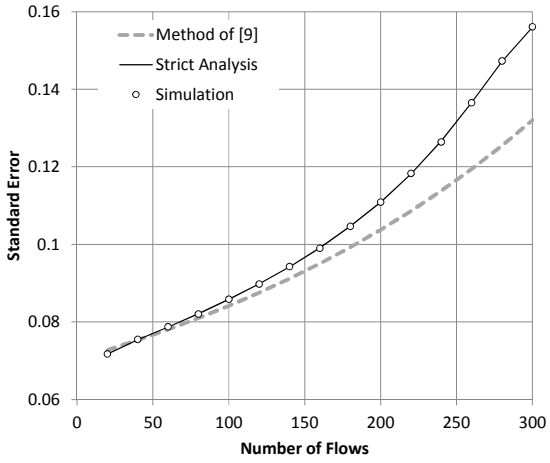

Fig. 6. Standard error for a small vector size: m = 101.

However, for $m = 10007$, the accuracy of the approximation, depending on the number of flows, is not always as good as in Fig. 7.Fig.8 plots the standard error obtained for larger values of n by keeping m to 10007. For this region of n, the method of [9] substantially underestimates the standard error. This may cause a problem in designing the vector size. Suppose that the standard error should be smaller than 5% and there exist 71500 flows. Then, the method of [9] will set m at 10007 to achieve the target

standard error because the error value obtained by (4) is 0.0497. However, the actual error value will be larger than 5%; the value is computed as 0.0567using the proposed strict analysis. Fortunately, the approximation does not greatly differ from the strict value. Thus, the problem caused by underestimation is avoidable by setting m to a slightly larger value than that obtained by (4).

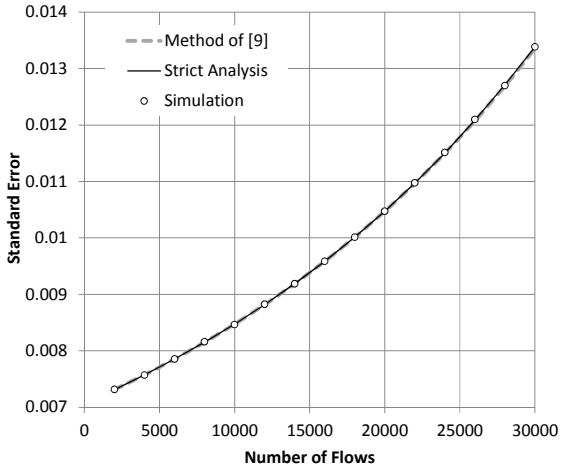

Fig. 7. Standard error for a moderate vector size: m = 10007.

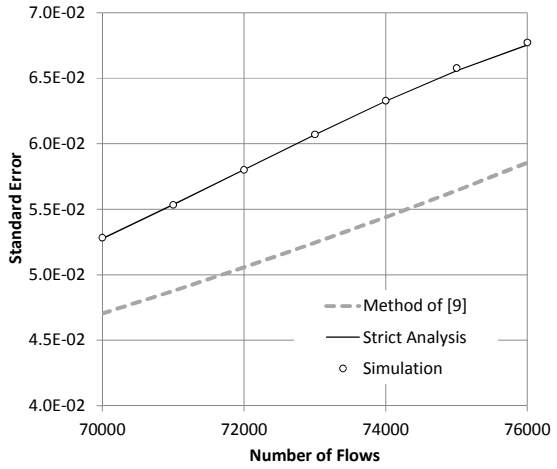

Fig. 8. Standard error for m = 10007 in the region where the method of [9] is not accurate.

Figs. 9 and 10 compare the proposed strict analysis with the method of [9] and the simulation result for the fill-up probability. Fig. 9 shows the characteristic for m = 101 while Fig. 10 shows that m = 10007. Fig. 9 shows, if m = 101, the method of [9] considerably overestimates the fill-up probability in comparison with the strict analysis and the simulation result. In contrast, Fig. 10 confirms that the approximation is in tandem with the strict analysis and the simulation result if m is as large as 10007.

For m = 10007, the approximate value of the fill-up probability is not very accurate if the fill-up probability is low.This is shown in Fig. 11, which compares the approximation with the strict analysis for the region where the fill up probability is less than 10^{-14}. In this figure, the simulation result is omitted because it is difficult to obtain reliable data with a sufficient number of fill-up events through the simulation. The figure shows that the approximation overestimates the fill-up probability. This means that the error by the approximation is on the safe side. That is, if m is determined for a target fill-up probability by using the formula of [9], the actual fill-up probability will be smaller.

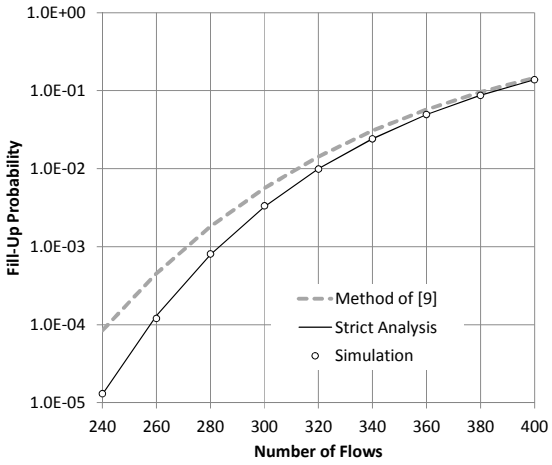

Fig. 9. Fill-up probability for a small vector size: m = 101.

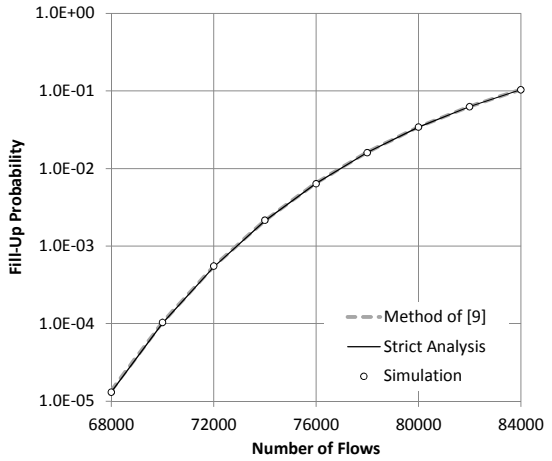

Fig. 10. Fill-up probability for a moderate vector size: m = 10007.

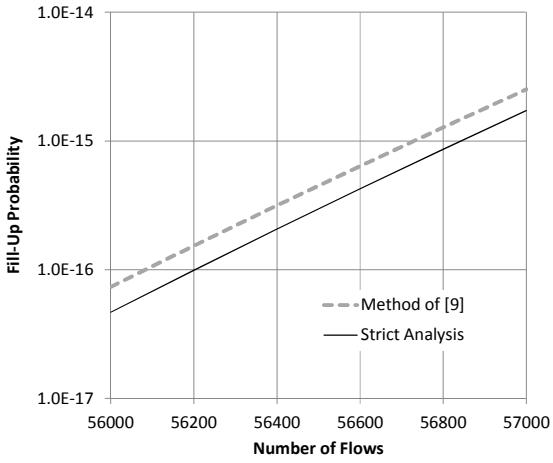

Fig. 11. Fill-up probability for m = 10007 in the region where the method of [9] is not accurate.

For practical flow counting, the expected flow number n will be considerably large. To estimate such a flow number accurately, the vector size m should also be large. Thus, the above results imply that the approximate formulas of [9] are considerably reliable for flow counting applications in real networks.

V. Design Issues in the Flow Counting Application

To apply the linear counting algorithm to flow counting, two design issues must be addressed. These are strategies for handling the vector fill-up problem and determining the appropriate vector size.

A. Vector Fill-Up Problem

If all vector elements are updated during the measurement period, U_n becomes 0. Thus, (2) does not yield any valid estimation. Therefore, it is necessary to establish a method to deal with this case. As such a method, [9] recommends rerunning the linear counting algorithm with a different hash function. This may be a practical solution for a database system, where the data is stored in a hard disk. Unfortunately, this method is inadequate for flow counting. To perform this method for flow counting, the flow identifiers of all arrived packets must be stored in the memory to prepare for possible re-execution of the algorithm. This requires an excessively large memory space. Moreover, if the algorithm is rerun, additional computational time is required to re-compute hash values and evaluate U_n. However, if a large memory space is available to store the flow identifiers, it is advisable to increase the vector size with the available memory space than to store the flow identifiers. Since a large vector size will make the fill-up probability negligibly small, it becomes unnecessary to store the flow identifier and rerun the algorithm.

This study proposes a very simple alternative method. That is, if the vector is filled up, the estimated flow number $\hat{n}$ is set to a constant,

$$\hat{n} = n_{\max}, \tag{19}$$

where $n_{\max}$ is the maximum number that the algorithm can evaluate with the vector size m. Understandably from (2), $n_{\max}$ is the estimated flow number when $U_n = 1$. Thus,

$$n_{\max} = m \log_e m . \tag{20}$$

With this method, it is unnecessary to store all the flow identifiers seen in the measurement period and rerun the algorithm with different hash functions. Thus, this method is very practical for real-time flow counting from the viewpoint of storage consumption as well as computational time. The method obviously outputs the most accurate estimation for $n > n_{\max}$. The method of [9] does not yield a solution that is larger than $n_{\max}$. This means that the estimation by the method of [9] is not better than the proposed method. For $n < n_{\max}$, the proposed method may overestimate the flow number. However, the expected error caused by this overestimation is not large. This is confirmed in Fig. 12.

Fig. 12 plots the simulation result that compares the standard error obtained by the proposed method with the method of [9]. The vector size m is 10007 in this figure. The figure shows that the error of the proposed method is slightly larger for $70000 < n < 86000$. However, the difference is not very large. For $n > 88000$, the error becomes smaller for the proposed method. This characteristic shows that the accuracy of the proposed method is not inferior to the method mentioned in [9].

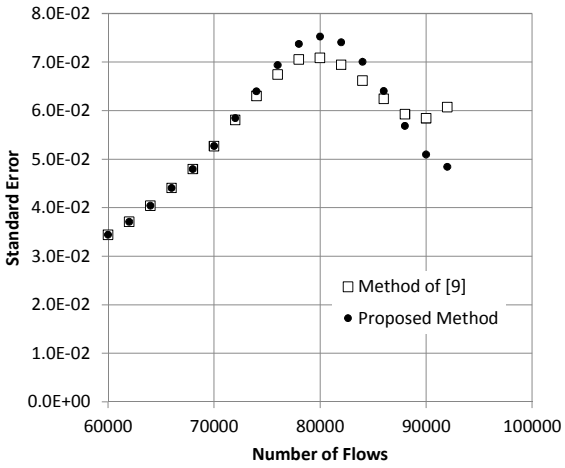

Fig. 12. Comparison between the methods that handle the vector fill-up problem.

B. Vector Size

The vector size should be determined to achieve a sufficiently low error and a negligibly small fill-up probability for the maximum number of flows. Thus, it becomes necessary to forecast the maximum number of flows observed in the measurement period.

The number of flows is bounded by the number of packets arriving in measured time. The number of packets is estimated by the product of the packet rate and time. The packet rate is bounded by the ratio of the link bit rate to the packet size. Therefore, for time T (s), the link bit rate r (b/s), and the minimum packet length $l_{\min}$, the flow number n is bounded as follows:

$$n \le \frac{rT}{l_{\min}} . \tag{21}$$

This upper bound is often not tight. However, it can be tight for extreme cases, for example, when the monitored link is under a UDP flood or a TCP SYN flood attack with a spoofed source address [13]. For these attacks, the link capacity may be fully used by short attack packets, each of which have a different source address and source port. Thus, the number of observed flow identifiers may approach that of the arrived attack packets.

For the open socket scenario, U_n is the number of vector elements untouched during the timeout period. Thus, the vector should not be fully used by the packets that arrive during the timeout period. This means that the right side of (21) must be evaluated by setting T to the timeout period. It should be noted that the timeout period may be larger than the measurement interval. For the active flow scenario, T is simply the interval from the start time t_1 to the stop time t_2. For example, assuming that the number of flows is estimated every 1 s, r is 1 Gb/s and $l_{\min}$ is 42 octets. Then, (21) concludes that n is not larger than 2.97×10^6. This requires the vector size m to be 3.84×10^5 if the standard error is less than 0.01.

VI. Evaluation by Network Data

The effectiveness of flow counting, which is based on the linear counting algorithm, is evaluated by using real-world network data. The evaluation is performed for two different scenarios. It is shown that the error characteristic is very different depending on the scenario.

A. Active Flow Scenario

The linear counting algorithm is implemented as a program that calculates the number of flows for the active flow scenario. The program can read a live packet stream as well as a `tcpdump`-format file through the `pcap` library [14]. The program is written in C language and runs on a Linux OS.

The program was executed for real-world network data, which is available from the MAWI database supported by the WIDE project [15]. From the files provided by the database, a 1 hour file was created by combining four 15-minute files taken on April 13, 2011 at sample point F. The input data file was then created by extracting IP version 4 TCP packets from this 1 hour file. The input data file was fed to the program, which estimated the number of flows with an interval of 1 s. For comparison purposes, the true number of flows was also obtained by the method described in Appendix B by using the `tcpslice` [14] and `tcptrace` [16] programs.

The input data was taken for a 150 Mb/s bidirectional link. From this bit rate, the upper bound of n is calculated as 8.9×10^5 by using (20). The actual number of flows was much smaller than this bound. Fig. 13 shows the output of the linear counting algorithm for $m = 10007$ in comparison with the true value. The output of the linear counting algorithm is very close to the true value. Fig. 14 depicts the close-up of the characteristics for $m = 50021$ and $m = 3001$. The figure indicates that the estimation for $m = 50021$ is very accurate. By contrast, a substantial error is observed for $m = 3001$. This shows that the statistical error of the algorithm is larger for a smaller value of m.

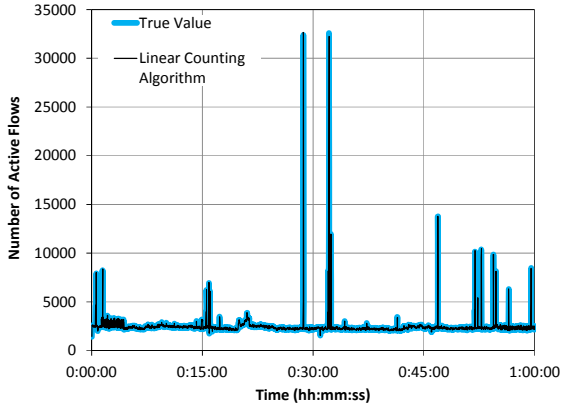

Fig. 13. Flow number evaluated by the linear counting algorithm in the active flow counting scenario.

To clarify the relationship between m and the statistical error, the standard error was evaluated for the linear counting algorithm by changing the value of m. This result is depicted in Fig. 15. The figure also shows the theoretical value of the standard error computed for $n = 2445$, which is the average flow number of the 1hour data file. Fig. 15 shows that the error decreases by increasing the value of m. In Addition, the error of the linear counting algorithm is very close to the theoretical standard error, which is obtained assuming $n = 2445$. This characteristic implies that the error is almost determined by the statistical nature of the algorithm for the active flow scenario. Thus, it is easy to exactly estimate the error for the vector size and the average flow number by using (4) or (13)–(16).

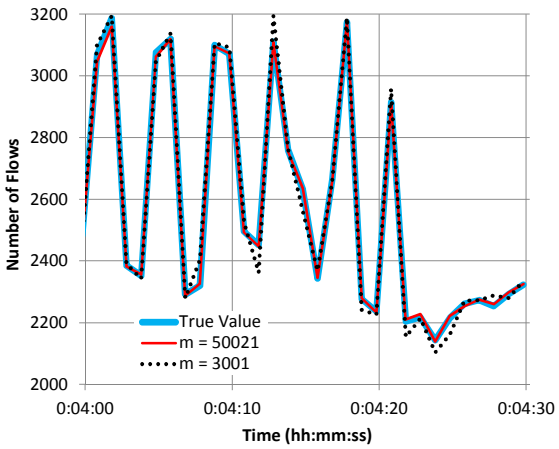

Fig. 14. Estimation by the linear counting algorithm for $m = 50021$ and $m = 3001$.

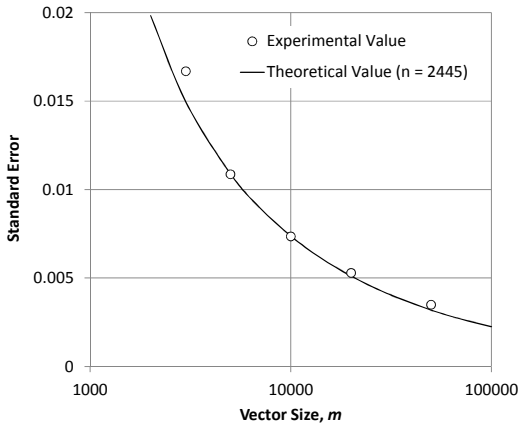

Fig. 15. Relationship between the standard error and the vector size for the active flow counting scenario.

The above results conclude that the number of flows is exactly estimated by the linear counting algorithm for the active flow scenario. Additionally, the actual estimation error is easily forecasted by the theory. Therefore, it is not difficult to determine the vector size for a given standard error, the fill-up probability, and an expected flow number for this scenario.

B. Open Socket Scenario

The linear counting algorithm was also examined for the open socket scenario. A program was implemented for this purpose as well. The program basically detects the termination of a flow by detecting timeouts. Thus, the program employs a timestamp vector (TV), which was introduced by [7]. However, the termination detection is based on a timeout, thus not very accurate. This means that a large estimation error is unavoidable. Thus, the program

employs three improvement techniques, which are suggested or examined in the literature [7], [10]–[12]. These techniques are referred to as T1, T2, and T3 hereafter and are described as follows.

T1) FIN/RST message utilization [7], [10]

This mechanism avoids the overestimation introduced by considering terminated flows to exist during the timeout period. Since a TCP flow is terminated with a FIN or RST message in a regular operation, its termination is basically discovered by watching a FIN or RST message. On the basis of this concept, the overestimation is eliminated by subtracting the number of flows that issued FIN/RST messages for the timeout period from the estimated flow number. The number of flows that issued the FIN/RST messages is easily counted by using another TV, whose elements are updated at arrivals of FIN/RST messages. This additional TV is referred as the FIN vector hereafter.

T2) Short timeout for one-packet flows [11], [12]

In real-world network data, there are many one-packet flows, each issuing only one packet (for example, a TCP SYN message). For one-packet flows, the termination cannot be detected by the FIN/RST messages and thus must be found by analyzing timeouts. Unfortunately, the timeout based method considers a one-packet flow to exist for the timeout period though it is actually terminated in a very short transmission time, causing excessive overestimation. This overestimation decreases by counting how many times each TV element is updated. Suppose that the update count of an element is 1. For this case, the element is obviously updated by one flow and only one packet has been issued from the flow. Then, it is likely that this flow is a one-packet flow. Thus, the overestimation is reduced by applying a smaller timeout period to such a vector element. Let u_h denote how many times the h-th vector element ($0 \le h \le m-1$) was updated. In the following, let $T_{o,1}$ denote the timeout period for the TV element having u_h = 1. Similarly, the timeout period is denoted by $T_{o,2}$ for the element having u_h = 2. Timeout period for other elements is T_o. The value u_h is reset to 0 when the measurement starts or the element is not updated for T_o.

T3) FIN/RST message count [11]

The accuracy of termination detection is improved by checking FIN/RST messages. However, a host may transmit multiple FIN messages repeatedly, if the peer host does not respond. In this case, the program considers the flow to be terminated by the first FIN message, though it is actually ended with the last FIN (or RST) message. If this happens, the flow number is underestimated because the flow is considered to end earlier than its actual termination. This underestimation is improved by counting the number of the FIN messages associated with the h-th vector element. Let v_h denote the number of FIN messages. Then, if v_h> 2, the h-th FIN vector element is not used to count the terminated flows. That is, the program considers that the flow associated with this element is repeatedly issuing multiple FIN messages and is not terminated. The counter v_h is reset to 0 when the measurement starts or the TV element is not updated for T_o. For an RST message arrival, v_h is set to 1 and thus the associated FIN vector element is utilized to count the terminated flows.

The improved version of the TV algorithm that employs the above techniques T1, T2, and T3 was implemented as a program, which was written in C language and runs on a Linux OS. It uses the `pcap` library to read live traffic as well as the `tcpdump` format files.

The program was executed for the same 1hour file that was used in the Section VI.A. Fig. 16 shows the result. In Fig. 16, m was set to 4000037. The timeout periods T_o, $T_{o,1}$, and $T_{o,2}$were set at 96 s, 1 s, and 10 s, respectively. These timeout values were chosen to achieve the best result.

By comparing Fig. 16 and Fig. 13, it is noticeable that the number of flows greatly differs for the two scenarios. The average flow number was 2445 for the active flow scenario while it was 7313 for the open socket scenario. This shows that the real-world network holds many low-rate inactive flows, which are not counted for the active flow scenario. This difference suggests that these two scenarios are completely different and should not be confused.

In Fig. 16, the output of the improved TV algorithm is considerably close to the true value. The standard error of estimation by the program was 0.046. Meanwhile, the theoretical error value of the linear counting algorithm is 3.5×10^{-4} for$m = 4000037$ and $n = 7313$. Therefore, the observed error is much greater than the theoretical value, computed by the statistical nature of the linear counting algorithm. In Addition, the error for the open socket scenario greatly depends on the parameters used in the termination detection. For example, if T_o increases to 110 s, the error increases to 0.076. Similarly, if T_o decreases to 80 s, the error increases to 0.088. Judging from these characteristics, it is concluded that the termination detection mechanism is the main cause of the estimation error for the open socket scenario.

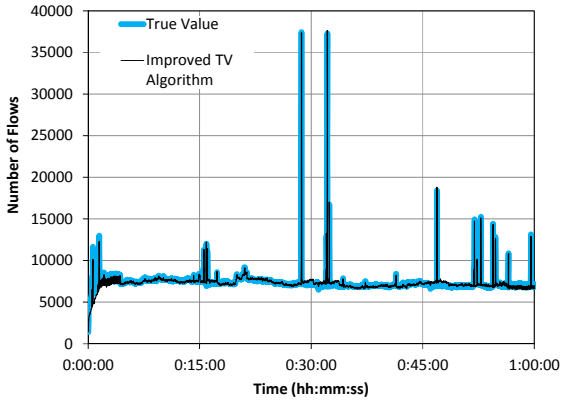

Fig. 16. Flow number evaluated by the improved TV algorithm in the open socket scenario.

The improvement techniques are essential to achieve good estimation accuracy. Fig. 17 shows the effectiveness of the technique T2. The timeout period was 4 s for the original TV algorithm, while T_o, $T_{o,1}$, and $T_{o,2}$were 11 s, 1 s, and 10 s, respectively, for the method improved with T2.These timeout values were chosen to minimize the standard error. In the given figure, the number of flows exhibits a peak from 0:28:41 to 0:28:42 because of temporally increased one-packet flows. For this period, the output of the original TV algorithm greatly overestimates the flow number. In contrast, the overestimation is completely removed by employing the technique T2. Fig. 18 compares the case of

employing only T2 with that of employing T1, T2, and T3. The figure shows that the accuracy is further improved by using T1 and T3 in addition to T2.

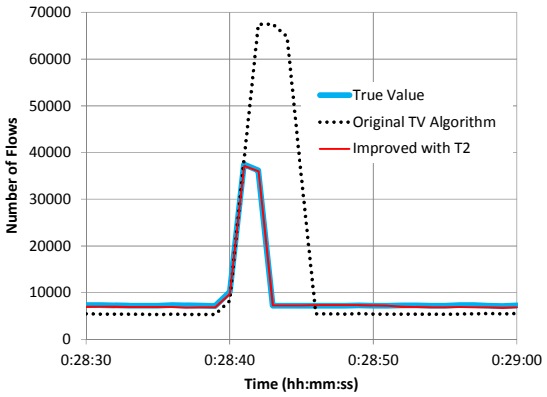

Fig. 17. Effectiveness of the improvement technique T2.

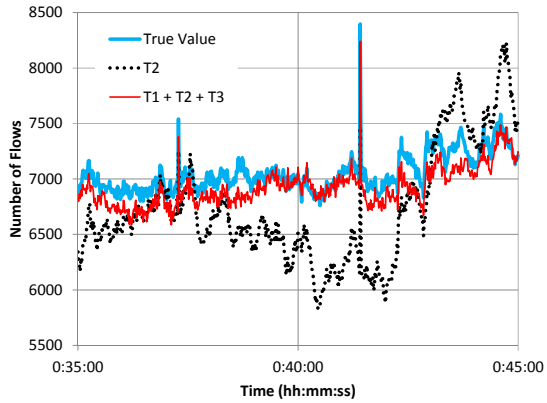

Fig. 18. Effectiveness of the improvement techniques T1 and T3.

The estimated accuracy of the improved TV algorithm greatly depends on the vector size m. For example, the standard error increases from 0.046 to 0.35 by reducing m from 4000037 to 500011. Fig. 19 compares the outputs for $m = 4000037$ and $m = 500011$. The figure shows that the flow number is overestimated for $m = 500011$. This overestimation is caused by T2 and T3. For these techniques, the counter values u_h and v_h exactly show the number of packets issued by a flow only if there is no collision for a hash value h. For T2, if two or more one-packet flows generate the same hash value, u_h will be greater than 1. Thus, the short timeout period $T_{o,1}$ is not applied to these one-packet flows. This causes overestimation. Similarly, for T3, if three flows generate the same hash value, v_h becomes larger than 2. Thus, the algorithm considers these flows to be retransmitting FIN messages. If this happens, the termination is not detected by the FIN messages. This also causes overestimation. Therefore, the collision probability among hash outputs must be negligibly small, for techniques T2 and T3 to work correctly. This requires a large vector size.

In conclusion, it is more difficult to accurately count the flow in the open socket scenario compared to the active flow scenario. This is because the error is caused by the detection of flow termination. Nevertheless, the estimation is considerably accurate as shown in Fig. 16, if the improvement techniques T1, T2, and T3 are employed. However, the vector size m must be large to obtain the accurate estimation with these techniques. This means that the required memory consumption is large for the open socket scenario.

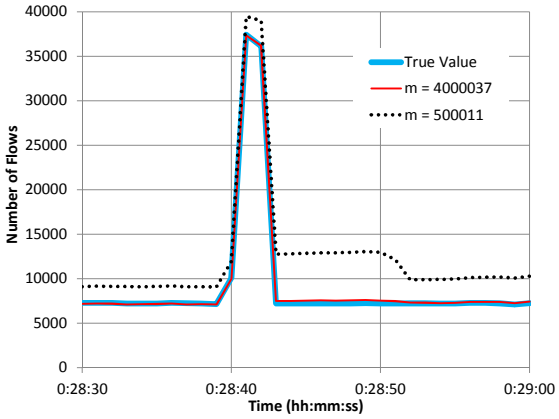

Fig. 19. Characteristic of the improved TV algorithm for $m = 4000037$ and $m = 500011$.

VII. Conclusion

This paper discusses the real-time flow counting technique based on the linear counting algorithm, which is based on a hash function and a vector. First, the paper proposed a strict analysis to clarify the exact statistical nature of the algorithm. The accuracy of the proposed analysis was confirmed through a computer simulation. Thereafter, strict analysis was compared with the approximate analysis derived in [9]. As a result, it was shown that the approximations were accurate with a few exceptions.

Next, the paper investigated how to treat the vector fill-up problem in flow counting. It was shown in the method of [9] that re-executes the algorithm with a different hash function is not adequate for flow counting. Instead, the paper reviewed a very simple method, which uses the maximum output value for the vector fill-up case. The simulation results confirmed the extensiveness of the method. The upper bound of the flow number was also investigated as a basis for determining the appropriate vector size.

This paper strictly distinguishes between two different measurement scenarios of flow counting: the active flow scenario and the open socket scenario. The algorithm was tested for these scenarios using real-world network data. As a result, for the active flow scenario, it was found that the estimation is accurate and the error is caused by the statistical nature of the algorithm. It is more difficult to accurately estimate the flow number for the open socket scenario. For this scenario, the estimation error is larger because of the difficulty in finding flow termination. It was also found that a large vector size is necessary for an open socket scenario. Estimating the flow number accurately with a small memory space remains to be an open problem.

Appendix

A. Hash Function

This study employs a prime-modulo-based hash function.

Assume that we are monitoring an IP version 4 packet stream. Let i_s and i_d denote the source and destination addresses, and j_s and j_d denote the source and destination port numbers, respectively. In addition, let p be the protocol field value. Then, for the flow identifier $x = (p, i_s, i_d, j_s, j_d)$, the function $h(x)$ is

$$h(x) = \{a\,(2^{16}\,p \oplus i_s \oplus i_d) + b\,(j_s \oplus j_d)\} \bmod m\,, \qquad \text{(a.1)}$$

Where a and b are constants and m is a prime number. In the simulations, a and b were set at 1. In Fig. 12, for the method of [9], b was changed to 1, 2, … to obtain different hash functions. For the evaluation that uses real-world network data, a and b were set to 253 and 31, respectively to obtain better results.

B. True Number of Flows

For active flow and open socket scenarios, the true number of flows was estimated as follows.

To obtain the number of active flows, the packet dump file was first sliced into 1 s files by the `tcpslice` [14] program. Each 1 s file is inputted into the `tcptrace` program. The output shows how many flows issued packets during a 1 s period.

For the open socket scenario, the packet dump file is first entered in the `tcptrace` program. The output of the program is stored in a text file. This output file contains the arrival times of the first and last packets in a flow. Assume that the flows are repeatedly counted at times t_1, t_2, …. Then, the true number of flows at t_i (i = 1, 2 …) are obtained by counting the flows of the first packet arriving before t_i and the last packet arriving after t_{i-1}. A Perl script was written to extract time information from the `tcptrace` output file and count the number of flows to be measured at t_i.

REFERENCES

[1] K. C. Claffy and H. W. Braun, "A parameterizable methodology for Internet traffic flow profiling,"*IEEE J. on Selected Areas in Communs.*, SAC-13, 8, 1995, pp. 1481-1494.

[2] 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.

[3] 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.*, E90-B, 11, 2007, pp. 3061-3072.

[4] 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.

[5] C. Estan, G. Varghese, and M. Fisk, "Bitmap algorithms for counting active flows on high-speed links, "*IEEE/ACM Trans. on Networking*, 14, 2006, pp. 925-937.

[6] 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.

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

[8] S. Zhu and S. Ohta, "Simple method to passively estimate the throughput of a TCP flow in IP networks," in *Proc. ICOIN 2010*, Busan, Korea, 2010, paper 5B-3.

[9] 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*, 15, 2, 1990, pp. 208-229.

[10] 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.

[11] S. Zhu and S. Ohta, "Real-time measurement of flows classified according to their application for IP networks, "*Cyber Journals: Multidisciplinary Journals in Science and Technology, Journal of Selected Areas in Telecommunications (JSAT)*, 2, 12, December Edition, 2011.

[12] S. Ohta and S. Zhu, "Real-time measurement of flows classified according to their application," in *Proc. APNOMS 2011*, Taipei, Taiwan, 2011, paper TS3-2.

[13] T. Peng, C. Leckie, and K. Ramamohanarao, "Survey of network-based defense mechanisms countering the DoS and DDoS problems, "*ACM Computing Surveys*, 39, 1, Article 3, 2007.

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

[15] *Wide: Working Group MAWI*, Available: http://www.wide.ad.jp/project/wg/mawi.html

[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 doctorate 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. Ohtais a member of the IEEE, IEICE, and ECTI. He received the Excellent Paper Award in 1991 from IEICE.