Self-Tuning the Parameter of Adaptive Non-Linear Sampling Method for Flow Statistics

Chengchen Hu[1,2], Bin Liu[1]
[1]Department of Computer Science and Technology
Tsinghua University
{huc,liub}@tsinghua.edu.cn
[2]State Key Laboratory of Networking and Switching Technology
Beijing University of Posts and Telecommunications

***Abstract*—Flow statistics is a basic task of passive measurement and has been widely used to characterize the state of the network. Adaptive Non-Linear Sampling (ANLS)is one of the most accurate and memory-efficient flow statistics method proposed recently. This paper studies the parameter setting problem for ANLS. A parameter self-tuning algorithm is proposed in this paper, which enlarges the parameter to a equilibrium tuning point and renormalizes the counter when counter overflows. It is demonstrated that the estimation error of ANLS with parameter self-tuning algorithm is improved by about 89 times for real trace, 70 times for Pareto traffic scenario and 370 times for exponential traffic, while giving the same memory size.**

***Index Terms*—Network measurement, flow statistics, counting, unbiased estimation**

I. Introduction

Network measurement provides many analyses including traffic matrix, flow volumes, flow size distributions, user sessions' duration and etc., which contributes to our capacity in network management and traffic engineering [5, 24]. Fundamentally, flow statistics is the main task of network measurement, which involves counting number of packets in terms of flows over a period of time [15]. A flow refers to a set of packets that have the same n-tuple value in the header fields.

With the continuous increase of Internet link speed and the number of flows, flow statistics has become a challenging task due to the demanding requirements on both memory size and memory bandwidth. Off-the-shelf memory is either low speed or low capacity. For instance, large capacity DRAM maybe large enough to hold all the flow records but its low speed disables the frequent accessing/updating rate to the counters, while fast SRAM supports high speed processing but is susceptible to overflow due to its limited memory capacity. Generally, there are two categories of solutions in the literature.

The first one sets full-size counters in DRAM, and its key problem is how to slow down the updates to the DRAM counters in order to match the I/O (Input/Output) speed of DRAM. Several combined SRAM&DRAM (SD) counter architectures fall in this category [21, 22, 25], whose basic idea is to store lower order bits of each counter in SRAM and all the counter bits in DRAM. SD solutions build a good architecture to set measurement counters, however, it has its limitations on 1) slow read access speed, 2) significant traffic over system bus, and 3) requirements of extra pin connections and board area.

The second one uses sampling technology to reduce the required counter size in order to keep the counters in SRAM [4, 8, 9, 12, 13]. The widely used static sampling (SS) method [4] selects packets with the same sampling rate/probability p for all the flows during the measurement interval. With SS, the probability of a flow with only one packet to be missed is much larger than the one of a flow with 10000 packets in the same link. Theoretical and experimental evaluation demonstrated that SS exhibits intolerably high relative error for small flows [13]. In our previous work [12, 13], we propose Adaptive Non-Linear Sampling (ANLS) method to adjust the sampling rate and control the estimation error of both large and small flows. It is demonstrated that ANLS is the most accurate flow statistics method given the same memory size compared with other related SRAM-based work [2, 7, 8, 10].

ANLS has a predefined control parameter a, which decides the measurement accuracy and counting range. Enlarging a decreases the counter overflow possibility, but decreases the measurement accuracy meanwhile. To avoid the possible overflow of the counters, a user would conservatively select a large a for large counting range and suffer from unnecessary accuracy sacrifice. In this paper, we study the performance variation of ANLS under different parameter settings and propose a parameter self-tuning method for ANLS during the measurement interval to strike the tradeoff between measurement accuracy and counting range.

The rest of the paper is organized as follows. Section II briefly reviews ANLS and describes the problem and idea of parameter tuning for ANLS. In Section III, we investigate the method to determine equilibrium point for parameter tuning. Section IV presents the counter renormalization processing. Section V gives the evaluations and Section VI describes the related work. Finally in Section VII, we conclude the paper.

This work is supported by NSFC (60625201, 60873250), the Cultivation Fund of the Key Scientific and Technical Innovation Project, MoE, China (705003), the Specialized Research Fund for the Doctoral Program of Higher Education of China (20060003058), 863 high-tech project (2007AA01Z216, 2007AA01Z468) and open project of state key Laboratory of Networking and Switching Technology (SKLNST-2008-1-05).

II. ANLS with Parameter Self-tuning

A. A brief review of ANLS

The main notations employed in this paper are illustrated in Table I. The counting process of original ANLS can be presented as $c \leftarrow c + 1$ with probability $p(c)$, where c is the counter value and $p(c)$ is calculated from the following equations with a pre-defined parameter $0 < a < 1$[1].

$$p(c) = \frac{1}{(1+a)^{c-1}}; \tag{1}$$

The sampling rate of ANLS $p(c)$ is adjusted according to the counter value and there is no need to predict or estimate the flow size distribution.

It is proven in [12, 13] that, the unbiased estimation of ANLS is $f(c)$ which is formulated in (2).

$$f(c) = [(1+a)^c - 1]/a. \tag{2}$$

With this estimation, the relative error[2] of flow size estimation (when the real flow size is n) can be presented as

$$e = \sqrt{(1 - 1/n)a/2}. \tag{3}$$

The relative error e converges to $\sqrt{a/2}$ when $n \to \infty$. It is also demonstrated that the counter value is tightly bounded by

$$f^{-1}(n) = \frac{\log(1+an)}{\log(1+a)}. \tag{4}$$

From (4), we observe that $f^{-1}(n)$ is an increasing concave function of real flow size indicating the scalable memory (counter) consumption of ANLS.

Although there are some literatures which study the adjustment of sampling rate to control the estimation error of small flows [2, 16], their input to tune the sampling rate is not accurate themselves. ANLS uses accurate counter value as the input to adjust sampling rate and derives the general theoretical principles on how to tune the sampling function. The most related work to ANLS is [18], which proposed a technique to count large number in small registers. However, ANLS gives a generic theoretical study on how to select sampling functions and present practical experiments to demonstrate the performance.

B. Problem statement

There are two major performance metrics for flow statistics. One is relative error and the other is counting range (related to memory consumption). Relative error measures the accuracy of a flow statistics method and can be quantified by coefficient of variation as shown in (3) for ANLS. Counting range is the largest flow size that a flow statistics method could record. The larger the counting range is, the more efficient the memory consumption is. For full size counter (like SD solution), the counting range is the largest counter value; and for ANLS, the counting range is

$$B = [(1+a)^c - 1]/a. \tag{5}$$

It is indicated by (3) that small a leads to small error, but it is also demonstrated by (5) that ANLS with small a has a small counting range and is easy to overflow the counters. The relative error curve and counting range curve are depicted in Figure 1 and Figure 2, respectively. Obviously, there is a tradeoff for ANLS between small relative error and large counting range by a configuration on parameter a.

In a practical flow statistics setting, the counter should not be overflowed during a single measurement interval. Therefore, a is always set to support the statistics for the largest possible flow size no matter what the actual largest flow size is. For instance, in the worst case, there can be about 31,250,000 packets for a OC-192 link in a one-second measurement interval, while our experiment shows that the actual largest flow in a real trace download form NLANR [19] has only 720,192 packets in a same interval. The conservative setting of a based on the theoretical largest flow size greatly degrades the accuracy performance. The degradation of relative error can be tens of times worse as demonstrated in Section V.

C. Parameter self-tuning algorithm

If we set the parameter a based on the actual flow size, the best and ideal tradeoff between accuracy and counting range would be achieved. Therefore, we study the method to adjust the parameter a during the statistic process in order

TABLE I
Table of notations

Notations	Descriptions
a	a predefined parameter of ANLS, $0 < a < 1$
c	the counter value
$p(c)$	the probability for counter update according to c
$f(c)$	unbiased estimation
n	the actual flow length
$\hat{n}$	the estimated flow length
e	coefficient of variation used for relative error evaluation
U_e	Accuracy Utility
U_b	Counting Range Utility

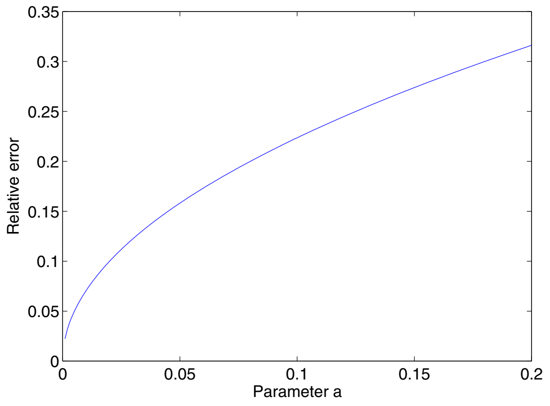

Fig. 1. Relative error vs. parameter a when flow size $n = 5000$.

[1]In this paper, we study a special case of ANLS as discussed in [12]. The conclusion can be easily extended to the general form of ANLS presented in [13].

[2]Actually, we use the coefficient of variation for relative error estimation.

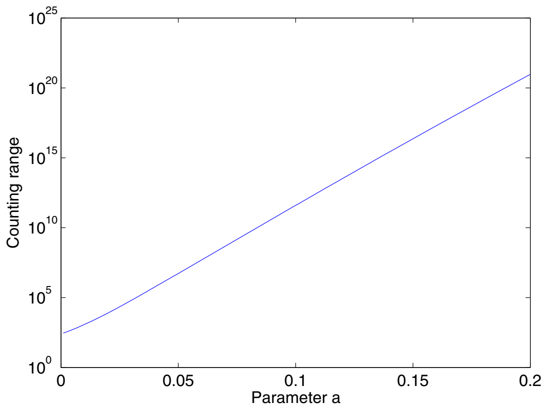

Fig. 2. Counting range vs. parameter a when the largest counter value is 256 (the counter is 8-bit width).

to provide the best and ideal tradeoff between accuracy and counting range. We patch the original ANLS method by adding a parameter self-tuning algorithm. The pseudo-code of ANLS with self-tuning algorithm is described in Algorithm 1.

Algorithm 1 ANLS with parameter self-tuning algorithm

```
$a_1 = 0$;
while a packet comes do
    extract the flow id. $i$ the packet belongs to;
    update corresponding counter $i$ using ANLS with parameter $a1$;
    if counter overflows then
        //self-tuning algorithm
        calculate the adjustment of parameter $a_2$;
        $a_1 = a_2$;
        renormalize the counter value;
    end if
end while
```

As shown in Figure 1 and Figure 2, with the increase of parameter a, the relative error, as well as the counting range of flow size, is monotonously increasing. To provide precise accuracy for flow statistics with given counter size in the beginning, the parameter a is first set to be zero (so $p(c) = 1$). In this way, ANLS counts every incoming packet and has no error. When the counter overflows, a is adjusted from $a_1$to a larger value a_2 and the counter is renormalized to a smaller value according to the reconfigured a_2. The further update of the counter will be according to ANLS with the new parameter a_2. The renormalization of the counter value and the reconfiguration of a_1 to a_2 is triggered whenever the counter overflows. The inverse estimation of the flow size[3] can be obtained from (2) based on the latest a. To implement the self-tuning algorithm, we need to solve the following two problems:

- What value should the parameter adjust to, *i.e.*, how to calculate a_2 in Algorithm 1? In Section III, we proposed a method to adjust parameter by seeking for the equilibrium tuning point.
- How to renormalize the counter when the counter overflows according to a_1 and a_2? The renormalization should keep the inverse estimation accurate and the method is described in Section IV.

III. Equilibrium Tuning Point

In this section, we study how to determine the equilibrium point for tuning the parameter when the counter overflows. We first define *Accuracy Utility* (U_e) and *Counting Range Utility* (U_b) during the parameter adjustment.

Suppose the parameters before and after renormalization are a_1 and a_2, respectively. The *Accuracy Utility* is defined as the following equation, where E_{max} is the maximum tolerant relative error[4].

$$U_e = \frac{E_{max} - e_2}{E_{max} - e_1}; \tag{6}$$

$$e_1 = \sqrt{(1 - 1/n)a_1/2}; \tag{7}$$

$$e_2 = \sqrt{(1 - 1/n)a_2/2}. \tag{8}$$

e_1 and e_2 are the relative error before and after the parameter tuning. e_1 is always larger than e_2 and thus U_e is between zero and one.

The *Counting Range Utility* U_b is defined as

$$U_b = \frac{B_2 - B_1}{B_2}; \tag{9}$$

$$B_1 = [(1 + a_1)^c - 1]/a_1; \tag{10}$$

$$B_2 = [(1 + a_2)^c - 1]/a_2. \tag{11}$$

B_1 and B_2 are the counting range before and after the parameter tuning. B_2 is always larger than B_1 and thus U_b is between zero and one too.

Before the parameter tuning, the accuracy utility is one and the counting range utility is zero. As shown in Figure 3, with the increase of parameter, the counting range utility increases and the accuracy utility decreases. When the two utility curves meets, the cross point is the equilibrium tuning point where we aim to adjust the parameter. The reason to select this equilibrium point as the tuning point is that the tuning of the parameter keeps the ANLS in a stable state, which achieves balance between accuracy metric and counting range metric.

$$U_e = U_b \tag{12}$$

Figure 3 is an illustration of the calculation of equilibrium tuning point when $a_1 = 0$. Given the counter width, we can use bisection method to pre-compute the equilibrium tuning point as shown in Algorithm 2. Instead of recording the actual value of tuning point associated with the counter, a further 4-bit identification is associated with each counter to indicate the value of a. When the counter overflows, the identification is increased by one. Table II illustrates the computation results using Algorithm 2, when 12-bit counters are utilized.

[3]The inverse estimation of the flow size is c if $a = 0$ in the end of the measurement interval

[4]The relative error should not exceed this threshold. We make $E_{max} = 0.1$ in this paper.

Algorithm 2 Finding the tuning point

```
float ETP(float a1, int c)
{
a2=0;bound=0.01;up=0.1;down=a1;
while abs(f)>bound do
    a2=(up+down)/2;
    n=((1+a1)^c-1)/2;
    e1=sqrt((1-1/n)*a1/2);
    e2=sqrt((1-1/n)*a2/2);
    Ue=(Emax-e2)/(Emax-e1);
    B1=((1+a1)^c-1)/a1;
    B2=((1+a2)^c-1)/a2;
    Ub=(B2-B1)/B2;
    f=Ub-Ue;
    if f>0 then
        down=a2;
    else
        up=a2;
    end if
end while
return a2;
}
```

TABLE II
TUNING POINT

Identification	Parameter	Counting range	relative error
0000	0.0006	11,874	0.0168
0001	0.0011	77,708	0.0239
0010	0.0017	472,359	0.0289
0011	0.0022	2,853,806	0.033
0100	0.0027	16,331,666	0.0364
0101	0.0031	86,998,135	0.0394
0110	0.0035	427,246,130	0.042
0111	0.0039	1,868,747,977	0.0443
1000	0.0043	6,860,213,041	0.0463
1001	0.0046	21,271,805,015	0.0479
1010	0.0048	49,831,262,588	0.0492
...			

IV. Renormalization

Again, suppose the counter value and the parameter right before parameter tuning are c_1 and a_1, respectively. Since the parameter tuning is only activated when the counter overflows, $c_1 = c_{max}$, where c_{max} is the maximum counter value according to the given counter width. The unbiased estimation of the flow size could be obtained from

$$\hat{n}_1 = f(c_1) = \frac{1}{a_1}[(1+a_1)^{c_1} - 1]. \tag{13}$$

The renormalization changes the parameter from a_1 to a_2, and correspondingly, the counter value is decreased from c_1 to c_2. The estimation from c_2 is as the following:

$$\hat{n}_2 = f(c_2) = \frac{1}{a_2}[(1+a_2)^{c_2} - 1]. \tag{14}$$

Please note that the principle of the renormalization is to keep the inverse estimations before and after the parameter tuning as the same. Namely, (13) and (14) should provide the same estimation. Therefore, we have,

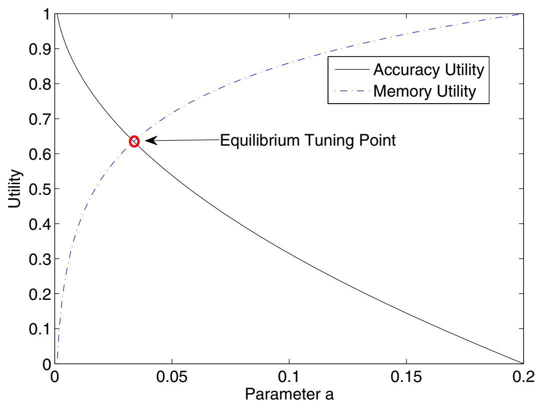

Fig. 3. Finding the equilibrium tuning point.

$$\frac{1}{a_1}[(1+a_1)^{c_1} - 1] = \frac{1}{a_2}[(1+a_2)^{c_2} - 1]; \tag{15}$$

$$c_2 = \log\{\frac{a_2}{a_1}[(1+a_1)^{c_1} - 1] + 1\}/\log(1+a_2). \tag{16}$$

However, the calculation from (16) is not always integer and we normalize the counter value with probability updates as shown in Algorithm 3. Let $X = \log\{\frac{a_2}{a_1}[(1+a_1)^{c_1} - 1] + 1\}/\log(1+a_2)$. The counter is renormalized to $\lceil X \rceil$ with probability $x - \lceil X \rceil$, and is reset to $\lfloor X \rfloor$ with probability $1 - (x - \lceil X \rceil)$.

Algorithm 3 Renormalization

```
// The counter overflows to trigger a renormalization
$X = \log\{\frac{a_2}{a_1}[(1+a_1)^{c_1} - 1] + 1\}/\log(1+a_2)$;
$v = rand(0,1)$; //A random variable between 0 and 1
if $v \le (X - ceil(X))$ then
    //ceil(X) rounds X to the nearest integer towards infinity
    $c = ceil(X)$;
else
    $c = ceil(X) - 1$;
end if
```

V. Evaluation

In this section, we first prove that the renormalization process does not introduce any error and ensure that the estimation after renormalization is the same as the one before renormalization.

Theorem 1: The expect error in renormalization process is zero.

Proof: Let $X = \log\{\frac{a_2}{a_1}[(1+a_1)^{c_1} - 1] + 1\}/\log(1+a_2)$ From the Algorithm 3, the expected value of c_2 can be formulated as

$$E(c_2) = \lceil X \rceil (X - \lfloor X \rfloor) + \lfloor X \rfloor (1 - X + \lfloor X \rfloor)$$

$$E(c_2) = X$$

Namely, $f(E(c_2)) = f(c_1)$.

TABLE III
RELATIVE ERROR WITH AND WITHOUT RENORMALIZATION

Para. tuning	No	Yes
Real trace	0.89%	0.01%
Pareto Scenario	3.58%	0.05%
Exp. Scenario	3.7%	0.01%

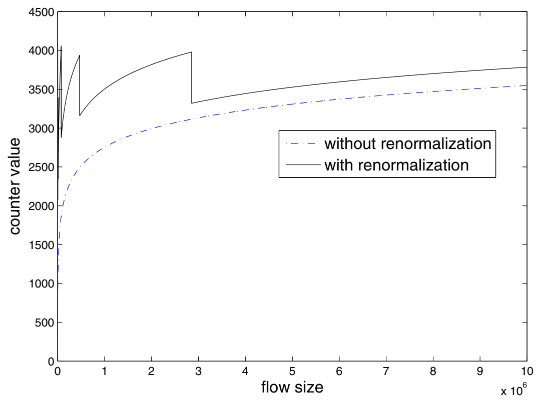

Fig. 4. Grow of the counter value.

■

Next, we compared the performance of ANLS with and without parameter tuning under real trace and synthetic traces. We first employ a real trace collected on an OC-192 link [19]. Furthermore, In order to examine the effects of flow size distribution, we generate synthetic data for a fully loaded OC-48 (2.5 Gbps) link within a one-minute measurement interval; therefore, the total number of packets in one minute is 468,750,000 in the worst case where all the packets are of 40 bytes. We first generate the flows whose sizes follow Pareto distribution (the shape parameter is 1.053 and the scale parameter is 4). We also synthesize data flows with an exponentially distributed size (the location parameter $\lambda = 500$, *i.e.*, the mean flow size is 500).

Table III illustrates the relative error of ANLS with and without parameter tuning under different traffic scenarios. In the experiment, the counter width for ANLS without parameter tuning is 16-bit and the counter width for ANLS with parameter tuning is 12-bit. Considering the 4-bit identification for a value, ANLS with or without parameter tuning consumes the same memory size. The results in Table III shows that the accuracy is improved by about 89 times for real trace, 70 times for Pareto traffic scenario and 370 times for exponential traffic, while giving the same memory size.

Figure 4 indicates the counter growth of ANLS with and without counter renormalization. In this experiment, the counter width is 12-bit, which means the largest counter value is 4096. As what we analyze above, the counter value of ANLS without parameter tuning is monotonously increasing while ANLS with parameter tuning efficiently use the counter resource by renormalizing the counter value when counter overflows. The frequency and scope of the renormalization is decreasing with the increase of flow size.

VI. RELATED WORK

Traditional counting system configures all the counters as the same size causing inefficient for flow length counting. Internet exhibits an "80-20" feature for its traffic pattern [20], *i.e.*, 80% of Internet packets are generated by 20% of the flows. Based on this observation, a recent work in [14] proposed BRICK (Bucketized Rank Index Counter) to organize efficient "variable-length" counters. The basic idea of BRICK is intuitive and is based on statistical multiplexing, which bundles groups of a fixed number (say 64) of counters that is randomly selected from the array, into buckets. BRICK allocates just enough bits to each counter in the sense that if its current value is c_i, BRICK allocates $\lfloor lg(c_i) \rfloor + 1$ bits to it. Besides, Counter Braids (CB) [17] is another novel counter organization for accurate flow measurement, which builds a hierarchy of counters braided via random graphs in tandem. CB allows the sharing of counter bits and thus the required counter bits are reduced. The motivations and gains of BRICK and CB are essentially different from the proposed solution in this paper. BRICK and CB reduce the total memory size by utilizing the features/relationship among all the counters, but they do not compress the size for each single counter. ANLS reduces the total memory size by compressing every single counter. In fact ANLS and BRICK/CB are complementary to each other and can be combined to get further reduction of counter size.

A combined SRAM&DRAM (SD) counter architecture is first proposed in [22]. A DRAM is utilized to store the full-size counters and a SRAM is employed to enable counter updates at line rate. The increments are first made only to SRAM counters, and the values of each SRAM counter is then committed to the corresponding DRAM counters before being overflow. The key problem in this architecture is how to design a Counter Management Algorithm (CMA) which determines the order of the SRAM counters to be flushed to DRAM counters. Different papers on the study of SD counter in fact investigate the CMA. In [22], Largest Counter First (LCF) algorithm is severed as CMA, which uses a heap-based priority queue to select the closest counter to be overflowed and then flush it to the DRAM. However, the SRAM memory size to maintain such a heap is about two times of the memory size to store the SRAM counters. A method to reduce the CMA complexity is further investigated in [21] by the use of an aggregated bitmap. In [25], the requirement on SRAM counter size and the control memory are further reduced. The authors in [25] employ a small write-back buffer to store indices of the overflowed counters and a simple randomized algorithm to statistically guarantee that SRAM counters do not overflow in bursts. The contributions of the SD solutions are significant for many application scenarios. However, SD architecture has its limitations. First, the read accesses of SD can only be done on DRAM side and thus is quite slow. Second, SD also significantly increases the amount of traffic between SRAM and DRAM across the system bus, which may lead to a serious bottleneck in system design [14]. Third, it is a trend to integrate measurement functions into routers; however, SD needs a dedicated SRAM and a dedicated DRAM, which will consume extra pins connections and board areas. It could be a big concern to implement it into an already crowded router line card.

A pioneering work on sampling of network traffic is published in [6], which uses static sampling for the purpose of measuring on the NSFNET backbone. Event and time driven sampling methods are studied in [6] to estimate statistics of distributions of inter-arrival time and packet size. The primary flow-level measurement tool used by network operators nowadays is NetFlow [3], which resorts to packet sampling (known as sampled NetFlow [4]), to handle the large traffic volume and diversity in high speed links. Considering the multi-hop feature of most flows, several attempts [1, 11, 23] are initiated to deploy the sampling system in a distributed manner for the purpose of passive measurement, which addresses the tradeoff between maximizing the measurement coverage of the network and minimizing the deployment cost (the lower sampling rate and less monitor beacons, the lower the deployment cost). The "sample and hold" method is introduced in [9], which uses a small and fast memory to process every packet in a real-time manner. This method is used to capture large flows but not for small flows. CATE is proposed in [10] to estimate the proportion of each flow by making multiple comparisons for each arrival and counting the number of coincidences. This method is accurate for media-size and large-size flows but is less accurate for small-size flows.

In the context of adaptive sampling, several mechanisms are introduced for different purposes. Better NetFlow (BNF) was proposed in [8] to improve memory utilization by an adaptive linear sampling method. A relatively large sampling rate is configured at the beginning of the measurement interval and will adaptively decrease when possible memory overflow is detected. A size-dependent sampling (SDS) mechanism was presented in [7]. A flow whose size is larger than z is always selected, while the flow with size $x < z$ is sampled with probability x/z. The authors in [2] provided an important theorem specifying the minimum number of packet samples required to guarantee the expected relative error, and they also proposed an adaptive random sampling (ARS) method. However, to utilize their theorem, it is required to first estimate the total packet amount using a linear auto-regressive (AR) prediction model. The accuracy and the implementation complexity of ARS are greatly restricted by the operations to determine the AR model parameters. All the above methods optimize on either the memory size or accuracy for medium to large flows, but can not guarantee the estimation error for small flows.

VII. Conclusion

In this paper, we study the parameter self-tuning mechanism for ANLS. ANLS has a predefined control parameter a, which decides the measurement accuracy and counting range. Enlarging a will decrease the counter overflow possibility and decrease the measurement accuracy meanwhile. A modification of ANLS has been proposed by adding a parameter self-tuning algorithm. The parameter a is first set as zero in order to ANLS counts every incoming packet without error. When the counter overflows, a is adjusted to a larger value and the counter is renormalized to a smaller value according to the reconfigured a. The further update of the counter will be according to ANLS with the new parameter a. The renormalization of the counter value and the reconfiguration of a is triggered whenever the counter overflows. Experiments under real trace and synthetic traces demonstrated that the proposed method improve the accuracy of ANLS tens of times.

References

[1] G. Cantieni, G. Iannaccone, C. Barakat, C. Diot, and P. Thiran. Reformulating the monitor placement problem: Optimal network-wide sampling. In *40th Annual Conference on Information Sciences and Systems*, pages 1725 – 1731, 2006.

[2] B.-Y. Choi, J. Park, and Z.-L. Zhang. Adaptive random sampling for load change detection. In *ACM SIGMETRICS 2002*, pages 272 – 273, 2002.

[3] Cisco. Cisco ios netflow data sheet. http://www.cisco.com.

[4] Cisco. Sampled netflow data sheet. http://www.cisco.com.

[5] K. Claffy and S. McCreary. Internet measurement and data analysis: Passive and active measurement. http://www.caida.org.

[6] K. C. Claffy, G. C. Polyzos, and H.-W. Braun. Application of sampling methodologies to network traffic characterization. In *ACM SIGCOMM 1993*, pages 194–203, 1993.

[7] N. Duffield, C. Lund, and M. Thorup. Learn more, sample less: Control of volume and variance in network measurement. *IEEE Trans. Inform. Theory*, 51:1756–1775, 2005.

[8] C. Estan, K. Keys, D. Moore, and G. Varghese. Building a better netflow. In *ACM SIGCOMM 2004*, pages 245 – 256, 2004.

[9] C. Estan and G. Varghese. New directions in traffic measurement and accounting. In *ACM SIGCOMM 2002*, pages 323 – 336, 2002.

[10] F. Hao, M. S. Kodialam, T. V. Lakshman, and H. Zhang. Fast, memory-efficient traffic estimation by coincidence counting. In *IEEE INFOCOM 2005*, 2005.

[11] C. Hu, B. Liu, Z. Liu, S. Gao, and D. O. Wu. Optimal deployment of distributed passive measurement monitors. In *ICC 2006*, volume 2, pages 621 – 626, 2006.

[12] C. Hu, S. Wang, B. Liu, and J. Tian. Control estimation error of sampling method for passive measurement. In *GLOBECOM 2007*, pages 2576–2580, Washington D.C., USA, 2007.

[13] C. Hu, S. Wang, J. Tian, B. Liu, Y. Cheng, and Y. Chen. Accurate and efficient traffic monitoring using adaptive non-linear sampling method. In *INFOCOM 2008*, Phoenix, USA, 2008.

[14] N. HUA, B. Lin, J. J. Xu, and H. C. Zhao. Brick: A novel exact active statistics counter architecture. In *ANCS 2008*, 2008.

[15] F. Khan, L. Yuan, C.-N. Chuah, and S. Ghiasi. A programmable architecture for scalable and real-time network traffic measurements. In *ACM ANCS'08*, 2008.

[16] A. Kumar and J. Xu. Sketch guided sampling – using on-line estimates of flow size for adaptive data collection. In *IEEE INFOCOM'06*, 2006.

[17] Y. Lu, A. Montanari, B. Prabhakar, S. Dharmapurikar, and A. Kabbani. Counter braids: A novel counter architecture for per-flow measurement. In *ACM SIGMETRICS*, 2008.

[18] R. Morris. Counting large numbers of events in small registers. *Commun. ACM*, 21(10):840–842, 1978.

[19] NLANR. Passive measurement and analysis (pma). http://pma.nlanr.net.

[20] K. Psounis, A. Ghosh, B. Prabhakar, and G. Wang. SIFT: a simple algorithm for trucking elephant flows and taking advantage of power laws. In *the 43rd Allerton Conference on Communication, Control, and Computing*, 2005.

[21] S. Ramabhadran and G. Varghes. Efficient implementation of a statistics counter architecture. In *ACM SIGCOMM'03*, 2003.

[22] D. shah, S. Iyer, B. Prabhakar, and N. McKeown. Maintaining statistics counters in router line cards. *IEEE Micro*, 22(1):76–81, 2002.

[23] K. Suh, Y. Guoy, J. Kurose, and D. Towsley. Locating network monitors: Complexity, heuristics, and coverage. In *INFOCOM 2005*, volume 1, pages 351–361, 2005.

[24] G. Varghese and C. Estan. The measurement manifesto. *ACM Computer Communication Review*, 34:9–14, 2004.

[25] Q. Zhao, J. J. Xu, and Z. Liu. Design of a novel statistics counter architecture with optimal space and time efficiency. In *ACM SIGMETRICS 2006*, 2006.