Adaptive Aggregation Flow Measurement on high speed Links

Guang Cheng
School of Computer Science & Engineering
Southeast University
Nanjing, P.R.China, 210096
gcheng@njnet.edu.cn

Jian Gong
School of Computer Science & Engineering
Southeast University
Nanjing, P.R.China, 210096
jgong@njnet.edu.cn

Abstract— **While network traffic may be characterized by many different criteria, it is ease to aggregate traffic along one dimension at a time. Unfortunately, by aggregating traffic along any single dimension, the network manager inevitably loses some interesting information. While the network manager can expose this structure by using finer grained representations, such as flows, he then must manage the excessive detail contained in such a representation. We define our traffic clusters in terms of the five fields typically used to define a fine-grained flow: source IP address, destination IP address, protocol, source port and destination port. Unlike others flow monitoring methods, such as NetFlow and ANF, we only keep the heavy-tailed flows and sampled short flows on a non-uniform sampling method with the flow length. The aggregation traffic can be estimated by these sampled flows and can keep the estimated accuracy at the same time. Experiment studies show our approach can significantly improve both the accuracy and efficiency in network aggregation flow monitoring comparing to other existing approaches.**

Keywords-Aggregation Flow, Adaptive Monitoring, Removal Flows

I. INTRODUCTION

Aggregation flow monitoring and analysis is crucial for many network applications, such as network planning, network management, and network security applications [1]. Network packets passing through the monitoring system can be classified into flows based on their header 5-tuple information, which can be further analyzed to present more significant implications. Aggregation functionality may turn useful in case of any anomaly (e.g., DoS attack, Worm spreading, etc) [2] where suddenly a lot of small flows are generated. NetFlow [3], implemented in Cisco routers, can generate and output flow records, and keep a flow cache into memory as flow records to describe the passing traffic. Current flow monitoring approaches, which require recording flow records into memory to keep the flow status, usually run into problems if the number of flows too huge to hold in the memory. Several approaches have been proposed to address this challenge. However they either lack flexibility in adapting to greatly varying network traffic (e.g. sNetFlow), or require intensive computing resources (e.g. ANF). Current countermeasures are (1) NetFlow: if the flow cache fill up, then depending on the defined behavior the result could be: 1a) if discarding new flows when the cache is full, legitimate new flows (even big ones) will not be accounted. 1b) if exporting aggressively non-terminated flows to make room for new ones, the other components in the system (e.g. collector) may get into trouble because of the flow record explosion. 2) Elephant flow detection: it will preserve "good" flows however the "bad" ones may pass unobserved. 3) Adaptive flow sampling: will ensure that the flow cache exporting rate remains stable, but accuracy in accounting legitimate flows will decrease anyway. Ant it brings complexity into the flow monitor.

In this paper, we propose a novel approach to tackle this issue by using an aggregation flow measurement (AFM) scheme to make the monitoring system self-adjustable to the varying monitored traffic. By caching estimating values rather than actual measured value, thus AFM can use multiple sampling rates to adapt the traffic change, and we can also keep the measured accuracy of each sampling rate and don't waste the measured resource during a high sampling rate. We also propose one non-uniform flow management policy that can control the flow cache size and keep the estimated accuracy at the same time.

The paper is organized as follows: we discussed the related work, and provide an introduction to NetFlow and ANF in Section 2. In Section 3, we elaborate our approach with detailed discussion on the corresponding algorithms. Comprehensive experiments are conducted and the results are discussed in Section4. Section 5 concludes this work and shows some potential improvement on this work.

II. RELATED WORK

With the increasing demands from various areas such as network security, network flow monitoring has gained more and more attentions by some research communities. The IETF Packet Sampling working group (PSAMP) [4] is chartered to define a standard set of capabilities for network elements to sample subsets of packets statistically. Chaudhuri [5] has proved that sampling can be compensated for the estimated traffic in packets or bytes. Kumar [6] proposed a novel SCBF that performs per-flow counting without maintaining per-flow state in and an algorithm for estimation of flow size distribution [7]. This is followed by [8] in which the flow distribution is inferred from the sampled statistics. Duffield [9] studied the statistical properties of packet-level sampling using real-world Internet traffic traces, and developed a simple model to predict both the export rate of flow packet-sampled flow statistics and the number of active flows. The measured numbers of flows

and the distribution of their lengths can been used to evaluate gains in deployment of web proxies [10], and determine thresholds for setting up connections in flow-switched networks [11]. Ribeiro [12] take a systematic approach to understand the contributions that different types of information within sample packets have on the quality of flow-level estimates. There are measured solutions for improving estimated precision of measured flow. Estan [13] proposed a sample and hold algorithms for identifying the large flows, and they also described the optimization of early removal further improve the accuracy. Ashwin Lall [14] also uses a similar sample and hold algorithm that after one item is sampled, an exact count is maintained for it. Raspall [15] present a Shared-State Sampling algorithm to detect a large of flows in the high-speed networks, which is a generation of sample and hold algorithm.

Cisco's NetFlow [3] is an open but proprietary network protocol developed by Cisco Systems to run on Cisco IOS-enabled equipment for collecting IP traffic information. In the case of NetFlow, Cisco uses the 5-tuple definition, where a flow is defined as a unidirectional sequence of packets all sharing all of the following 5 values: Source IP, Destination IP, source port, destination port, and protocol. Maintaining NetFlow data can be computationally expensive for the router and burden the router's CPU to the point where it runs out of capacity. To avoid problems caused by router CPU exhaustion, Cisco provides "Sampled NetFlow" Rather than looking at every packet to maintain NetFlow records, the router looks at every nth packet, where n can be configured or it is a randomly selecting interval. When using sampled Netflow records, we can estimate the result by multiplying n, but it will bring into some error for the sampling rate.

However Sampled NetFlow suffers from two problems: (1) the number of flow cached in system memory could be significantly increasing if the monitored network hit by certain burst traffic such as DDoS; (2) pre-selected sampling rate cannot be adapted to the varying network traffic. Estan [16] has proposed a family of bitmap algorithms for counting active flows and an adaptive NetFlow (ANF) was proposed. The adaptive NetFlow algorithm keeps resource within fixed limits, and uses renormalization to reduce the number of NetFlow records using a new sampling rate. This algorithm divides traffic stream into some fixed time interval bins. However, the algorithm has the following limitations: (1) it consumes a lot of CPU resource during renormalization because it needs repeatedly to analyze previously collected flow records that have already been processed and stored in system memory; (2) it inevitably loses monitoring accuracy due to static sampling rate; (3) it has to frequently adjust its sampling rate to adapt to varying traffic flow statistics.

All of these sampling techniques are valuable to study our algorithm. Obviously, there is a trade-off between monitoring accuracy and limited system resources (e.g. memory size, CPU speed). How to select an optimal sampling rate to achieve satisfactory accuracy with given system resource is a significant challenge. In this paper, we will try to tackle this issue.

III. ADAPTIVE AGGREGATION FLOW MEASUREMENT SYSTEM

Measured traffic must solve two problems: to improve estimated precision and to reduce measured resources. It is very difficult to configure the sampling rate to adapt the CPU and memory resource. We set two different sampling processes to control CPU resource and memory resource respectively in our AFM algorithm. A sampling process controls the number of packet through the CPU and consumption of CPU resource, and we name the sampling process as CPU sampling process, and call the sampling rate to CPU sampling rate. Packets through CPU will be aggregated into flows and recorded into a flow memory. The number of flow decides the size of flow memory, so we use another sampling process to sample these packets to control the number of short flow, and save the size of flow memory. We call this sampling process to memory sampling process, and name the sampling rate as memory sampling rate. If a flow is sampled into the flow memory, then its following packets are sampled that won't impart the size of flow memory, only changes the estimated precision of measured flow. The memory sampling process is flow sample and hold algorithm.

During the measuring traffic, if the traffic is increasing, and it is too heavy to have enough CPU resource, then the CPU sampling rate can be adjusted to a small sampling rate to adapt the traffic change. If the traffic is decreasing, the CPU has space resource to process more traffic, and then the CPU sampling rate can be increased to collect more traffic. Our algorithm can adjust the CPU sampling rate according to the traffic change, so multiple CPU sampling rates are used during the measured process. We record the estimated value when each packet is measured instead of the measured value directly, so the entry value in the flow memory is equal to the estimated value of measured flow.

The Flow Sample & Hold process (FSH) updates flow information by considering new received packets. FSH adopts a flow sample & hold mechanism, which records all packets information which belong to a existed flows, and then updates the corresponding flow entries. Otherwise, the packets are sampled with a probability and new entries will be created accordingly. In this manner, the heavy-tailed flows will be given higher priority, which also contain more information. Thus, this approach can maximize flow information and estimation accuracy with given flow cache size. The method takes removal policy which removes small flows to control the total number of flows in the flow cache.

We show the architecture of the system in the figure1. The architecture consists of three processes: CPU Sampling Process, Sample & Hold Process, and Removal Process. In the figure 1, all packets are sample by one out of n^{th}, and all sampled packets are processed by the sample & hold process. If there is a flow entry in the flow memory, then the flow entry is updated, otherwise the packet is sampled again using one out of m^{th} to decide whether a new flow is created. If the number of flows in the flow memory is larger than a threshold R, then a removal algorithm will be started up to remove some flow entries from the flow memory.

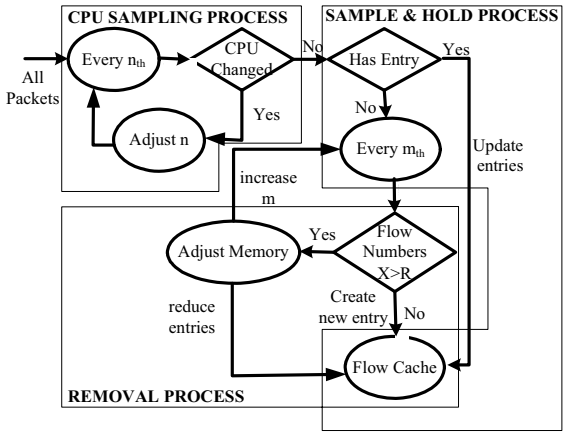


Figure 1. The Architecture of AFM

A. CPU Sampling Process

CPU sampling process is a random sampling process, and let sampling ratio p is equal to 1/n, which is 1 in n packets. A packet is sampled with probability p. Therefore to obtain an unbiased estimator $\hat{N}$ of the number of packets on the sample traffic, we should statistically compensate for the fact that with probability 1-p, the packet will miss due to the sampling process. It is intuitive that if we add 1/p=n to $\hat{N}$, the resulting estimator is unbiased. Suppose in a measurement epoch, we sample K packets with sampling ratio p_i, $\{ pkt_i, p_i, i = 1, \cdots K \}$. The output of estimated packets, which is an unbiased estimator of sampling traffic $\hat{N} = \frac{n}{p} = \sum_{i=1}^{n} \frac{1}{p_i}$. $1/p_i$ is an unbiased estimator of each measured value. If we use $1/p_i$ the measured packet, we can record the estimated values with different sampling ratio, and its standard deviation is $1/p_i$. According to the estimator $\hat{N}$, we can get the standard deviation of N, SD(N)= $\sqrt{\sum_{i=1}^{n} 1/p_i^2}$.

B. Update Flow Cache

We use a sample & hold algorithm to update flow cache. After a flow is recorded, all its follow packets are held. The memory sampling ratio is 1 in N, that is p = 1/N. supposed that a flow f is sampled, there are x packets which have been missed, when the first packet in flow f is sampled, so x obey geometric probability distribution, and its probability distribution is $p(1-p)^x$, so E(x)=1/p-1, D(x)=(1-p)/p^2. When the first packet in flow f is collected, 1/p packets have passed, so we use 1/p as the initial value of flow f records rather than 1. Its standard deviation SD(x)=sqrt(1-p)/p. As soon as the first packet in flow f is collected, all following packets in flow f will be recorded in flow f entry. Let the following packets is c, then the value in flow f entry is equal to c+1/p. So the standard deviation of flow f is SD(x)=sqrt(1-p)/p.

C. Remove Flow Entries from the Flow Cach

When the flow cache is full, we should remove some flow entries to have some spaces to accept the future flows. Long flow can record more packet information than that of small flow. If we can keep more long flows in the flow cache then we can get more accurate estimated value. So the removal policy is to keep more long flows and, on the other hand, we should keep some parts of small flows to estimate the remove small flows. These removal flows only contain small packets, so the estimated error of small flows can't affect the aggregation flows too much. In this paper, the flow cache is divided into the kept flow cache and the removal flow cache. We can configure more flow cache for the recorded flow cache than for the removal flow cache. Such as the recorded flow cache is set 95% of the total size of the flow cache, and removal flow cache is only set 5% of the flow cache size.

Long flow can record more packet information than small flow, so we remove all flows smaller than a threshold. When the number of flow entries in the recorded flow cache is over a predefined threshold B%M, where M is the size of flow cache, the removal process is triggered. After the removal process is over, the number of flow entries in the recorded flow cache needs less than F%M, so the number of removal flow entries is (B-F)%M at least. In the removal process, a computed threshold H assures that the number of flows smaller than H is over (B-F)%M. Let the length distribution of flow entries in the recorded flow cache be f(x), we can find a H, that $\sum_{i=1}^{H} f(x_i) \geq (B\% - F\%)M$, and at the same time $\sum_{i=1}^{H-1} f(x_i) < (B\% - F\%)M$. The system may take long time to remove a predefined number of flows if we have to go through the entire flow cache multiple times. Here we introduce a quick search algorithm. We set a k size dimension K which is updated if f the flow cache is updated. The index in the dimension K is the length of flow in the flow cache. If we want to set H as the removal threshold, according to the dimension K, the length of removal flow can be computed. If the k is too small, it may be $H > \sum_{i=1}^{k} K[i]$, then we can set d = k+1, and loop the flow cache to remove the threshold H flows.

Let the size of removal flow cache be K, the size of removal flows be L, and current sampling rate be 1 in N, the sampling rate 1 in N need to be adjusted to 1 in M to put both all removal flows just now and all flows in removal flow cache into the removal flow cache based on the new sampling rate 1 in M, M>N. Let the size of a flow in the removal flow cache be x. after flow x is sampled again by 1 in M, its values is $x \cdot N / M$. In the removal memory cache, a integer only is recorded, so sampling x with probability $\lfloor x \cdot N / M \rfloor$ is $\lceil x \cdot N / M \rceil - x \cdot N / M$, and sampling x with the probability $x \cdot N / M - \lfloor x \cdot N / M \rfloor$ is $\lceil x \cdot N / M \rceil$. The sampled value of a flow y in the removal flows from the recorded flow cache is $\lfloor y / M \rfloor$ with the probability $\lceil y / M \rceil - y / M$, and $\lceil y / M \rceil$ with probability $\lceil y / M \rceil$.

It costs a lot of CPU resource to compute a random value for each flow entry. In this paper, we give a simple method to compute the random value. Before the system begins, an initial

random value, "rand", is produced. Let the sampled value of a flow be z, z=y/M or $x \cdot N / M$, and its integer be $\lfloor z \rfloor$. If (z-$\lfloor z \rfloor$)+rand is larger than or equal to 1, then the sampled value is $\lfloor z \rfloor$+1, and one new random value is rand= (z-$\lfloor z \rfloor$)+rand-1. If (z-$\lfloor z \rfloor$)+rand is less than 1, then the sampled value is set to $\lfloor z \rfloor$, and the new random value is set to rand= (z-$\lfloor z \rfloor$)+rand. The new random value will be used in the next random computation. In this way, we need only to compute a random value by a random function at the initial period, and the following random value can be obtained by some addition and subtraction operations.

D. Performance Analysis

Let the size of flow cache M, F be the size of the recorded flow cache, and G be the size of the removal flow cache. F and G are also belong to M, and F+G<M. The algorithm also leaves M-(F+G) spaces to record the coming packets during the system removes some entries from the recorded flow cache. Such as F = 90%*M, G=5%*M, and we also have another 5%*M to record the coming flow during removal entries.

During removal flow entries, supposed a removal threshold H, if the size of a flow is less than H, then the flow will be removed, and renormalized to record into removal flow cache. A flow is larger than H, then it will be kept in the recorded flow cache. Let memory flow rate be p, and c is measured the count of flow s. 1/p can compensate for the missed packets, so the estimated packets of flow s is c+1/p, and its standard deviation is 1/p.

These removal flow entries in the recorded flow cache and these flow entries in the removal flow cache are renormalized and all sampled entries are recorded into the removal flow cache. Suppose the current sampling rate in the removal flow cache be p_{old}, and the new sampling rate be p_{new}. All removal flows from recorded flow cache are computed by $x*p_{new}$, and all flows in the removal flow cache are sampled again by $x*p_{new}/ p_{old}$.

Let the sampling rate which the first packet in flow s is sampled from sample & hold be p, c be the measured value, so the estimated value is x=(c-1)+1/p. If we sample these removal flows from the recorded flow cache randomly by sampling rate p_{new}, and re-sample all flows in removal flow cache by sampling rate p_{new}/ p_{old}. Let p_{new}/ p_{old} or p_{new} be q, then the sampling packet count obeys binomial distribution with mean qx, and variance q(1-q)x, so the new estimator is k=qx=q(c+1/p)=qc+q/p, and standard deviation is D(k)=q(1-q)c+q(1-q)/p^2. Let the sampled value be k, k/q be the estimator of x, so we can obtain the standard deviation in equation (1)

$$D(x) = \frac{q(1-q)c}{q^2} + \frac{q(1-q)}{p^2 q^2} = \frac{(1-q)c}{q} + \frac{(1-q)}{p^2 q} \quad (1)$$

Let m be the removal threshold in the recorded flow cache, so p<=1/m

$$D(x) \le \frac{(1-q)c}{q} + \frac{1-q}{(1/m)^2 q} < \frac{c}{q} + \frac{1}{(1/m)^2 q} < \frac{c \cdot (1/m)^2 + 1}{(1/m)^2 \cdot q}$$

$$D(x) < \frac{(1/m)+1}{(1/m)^2 \cdot q} = \frac{m+m^2}{q} \quad (2)$$

Let an aggregation flow z has f flows in the recorded flow cache, and n flows in the removal flow cache, and CPU sampling rate be p_{CPU}. We can obtain an estimator of the aggregation flows z in the equation (3), and its standard deviation in the equation (4).

$$\hat{z} = \sum_{i=1}^{f} x_i / p_{CPU} + \sum_{i=1}^{n} y_i / p_{CPU} p_{new} \quad (3)$$

$$m_z = \frac{1}{p_{CPU}} \sqrt{\sum_{i=1}^{f} 1/p_i^2 + \sum_{i=1}^{n} \frac{m+m^2}{p_{new}}} \quad (4)$$

Where p_i is the memory sampling rate when the first packet in flow i is sampled, but it is hard to remember all p_i. p_{cur} is the last memory sampling rate in the measuring epoch, so we know $p_i<=1/p_{cur}$. We can get an approximate standard deviation equation (5)

$$m_z = \frac{1}{p_{CPU}} \sqrt{\frac{f}{p_{cur}^2} + \frac{n(m+m^2)}{p_{new}}} \quad (5)$$

Let the size of every flow in the recorded flow cache be x_i, and we know $1/p_i<=x_i$. If $x_i < 1/p_{cur}$, then $1/x_i$ is closer p_i than p_{cur}. If $1/x_i$ is replaced as pi, then we can get more approximate value than the equation (5). Equation (6) is the more accurate standard deviation.

$$m_z = \frac{1}{p_{CPU}} \sqrt{\sum_{i=1}^{f} \frac{1}{\min(x_i^2, p_{cur}^2)} + \frac{n(m+m^2)}{p_{new}}} \quad (6)$$

IV. EXPERIMENTS

We use a group of packet traces gathered at NLANR [17], which uses OC192MON hardware to collect data on August 19, 2004, from 13:40pm to 14:40pm. These traces sketch is displayed in table 1. Estan's ANF algorithm has the same estimator precision with the Sampled NetFlow algorithm, except that Sampled NetFlow can't change its sampling rate, so in the experiments, we compare only ANF [16] with the AFM.

In the table 2, the first column ANF_0.1 is the measured error of ANF aggregation flows whose size is larger than 0.1% of total traffic, and AFM _0.1 is the measured error of AFM aggregation flow whose size is larger than 0.1% of total traffic. Both ANF_0.01 and AFM_0.01 are the aggregation flow larger than 0.01 of the total traffic. The first row 5% percentile is the error value which is located in the 5% percentile of all the errors. And 25% percentile, 50% percentile, 75% percentile, and 95% percentile all have the same meaning. The size of flow cache is set to 8192.

The table 2 shows us that the measured errors of AFM are better than that of ANF. Before we begin to examine the performance of different algorithms, we define one error metric. The $error_i$ metric in the equation (7) values the estimated error of the i_{th} aggregation flow, where X_i is the

actual packet numbers of the i_{th} aggregation flow, and $\hat{X}_i$ is the estimated value of the i_{th} aggregation flow.

Table 1. Traces Used in This Experiment

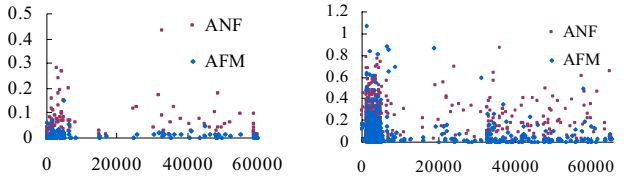
No.	Time	File Size	#Packets	#Flows
1	2004/08/19 13:40	173MB	8434885	144813
2	2004/08/19 13:50	154MB	6922629	146665
3	2004/08/19 14:00	172MB	8251311	184213
4	2004/08/19 14:10	157MB	7111907	154750
5	2004/08/19 14:20	161MB	7388868	140297
6	2004/08/19 14:30	176MB	8560571	143341
7	2004/08/19 14:40	162MB	7527445	149196

Table2. Error Comparison of Aggregation Flows

Percentile	5%	25%	50%	75%	95%
ANF_0.1	0.0044	0.0221	0.0514	0.0992	0.1975
AFM-0.1	0.0005	0.0027	0.0071	0.0161	0.0571
ANF_0.01	0.0157	0.0841	0.1830	0.3349	0.6364
AFM-0.01	0.0042	0.0213	0.0576	0.1302	0.4314

$$error_i = (X_i - \hat{X}_i) / X_i \times 100\% \quad (7)$$

The measured error of all application packets, which is larger than 0.1% of the total packets, is compared among the two algorithms in the figure 2(a), and errors which are larger than 0.01%, and less than 0.1%, are showed in the figure 2(b). In this experiment, we let flow cache be 8192, and experiment data is the first team data in the table 1, and only analyze the SPORT aggregation flows whose size is larger than 0.1% of the total packets. In the figure, X axis means SPORT number and Y axis in the two figures means relative estimated error. The figure 2 show that these points which are belong to AFM are closer X axis than the ANF algorithm, so the measured error of AFM is better than the ANF algorithm.



(a) (b)

Figure 2(a) Error of Aggregation Flows Larger than 0.1% Total Packets. Figure 2(b) Error of Aggregation Flows Between 0.01% and 0.1% Total Packets.

V. CONCLUSION

This paper presents an AFM method to detect flow information, which includes CPU sample process, flow sample & hold process, and flow removal process. We can control the flow number in the flow cache by the flow removal process, and sample & hold process can record more packets into the flow cache. In this paper, we propose one removal process, which removes small-flows directly from the recorded flow cache, and records removal flows into the removal flow cache. The removal process can be easy to manage the size of the flow cache, and estimate aggregation flow accurately. We use NLANR data to compare the performance the AFM algorithms with different removal process respectively, and the Estan's ANF algorithm, and analyze the SPORT aggregation flows over 0.1% and between 0.1%~0.01% the total traffic. The experiment shows that the AFM method has better precision than the ANF algorithm under the same flow memory size and configures, and the accuracy of the AFM algorithm is better than that of ANF algorithm with the same system resources.

ACKNOWLEDGMENT

This work has been supported by the National Grand Fundamental Research 973 program of China under Grant No. 2009CB320505, the Excellent Youth Teacher of Southeast University Program under Grant No. 4009001018, and the Free Research Program of Key Lab of Computer Network in Guangdong Province under Grant No. CCNL 200706, and the 2008 Natural Science Fundamental Program of Jiangsu Province under Grant No. BK2008288.

REFERENCES

[1] Xin Li, Fang Bian, Mark Crovella, etc. Detection and Identification of Network Anomalis Using Sketch Subspaces, IMC Oct. 2006

[2] Anukool Lakhina, Mark Crovella, Christophe Diot, Mining Anomalies Using Traffic Feature Distributions, In SIGCOMM, Aug. 2005.

[3] Cisco IOS NetFlow Introduction, http://www.cisco.com/en/US/products /ps6601/products_ios_protocol_group_home.html

[4] Packet Sampling (PSAMP), http://www.ietf.org/html.charters/psamp-charter.html, 2007.1.

[5] S. Chaudhuri, R. Motwani, and V. Narasayya. Random Sampling for histogram construction: How much is enough? In Proceedings of the ACM SIGMOD, 1998.

[6] Abhishek Kumar, Jun Xu, Li Li, and Jia Wang, Space Code Bloom Filter for Efficient Traffic Flow Measurement, ACM/USENIX IMC, Miami, FL, October 2003.

[7] Abhishek Kumar, Minho Sung, Jun (Jim) Xu and Jia Wang, Data streaming algorithms for efficient and accurate estimation of flow size distribution, ACM SIGMETRICS 2004.

[8] Duffield, N.G., Lund, C. , Thorup, M.: Estimating Flow Distributions from Sampled Flow Statistics. ACM SIGCOMM . 2003, Karlsruhe,Germany. August 25-29. 325-336.

[9] Duffield, N.G., Lund, C., Thorup, M.: Properties and Prediction of Flow Statistics from Sampled Packet Streams, ACM SIGCOMM IMW 2002, November 6-8, 2002.

[10] Feldmann, A. , Caceres, R. , Douglis, F. , Glass, G., Rabinovich, M.: Performance of Web Proxy Caching in Heterogeneous Bandwidth Environments, IEEE INFOCOM 99.

[11] Feldmann, A., Rexford, J., and C¡äaceres, R.: Efficient Policies for Carrying Web Traffic over Flow-Switched Networks, IEEE/ACM Transactions on Networking, vol. 6, no.6, pp.673-685, December 1998.

[12] Ribeiro, B., Towsley D., Ye, T., Bolot, J. Fisher Information of Sampled Packets: an Application to Flow Size Estimation, IMC' 06, Oct. 25-27, 2006, Rio de Janeiro, Brazil.

[13] C. Estan and G. Varghese. New directions in traffic measurement and accounting. In SIGCOMM, Aug. 2002.

[14] Ashwin Lall, Vyas Sekar, Mitsunori Ogihara, etc., Data streaming algorithms for estimating entropy of network traffic, ACM SIGMETRICS Performance Evaluation Review, Vol34 ,No.1, June 2006.

[15] Frederic Raspall, Sebastia Sallent, Josep Yufera, Shared State Sampling, in Proceeding of IMC Oct. 2006.

[16] C. Estan, Ken Keys, David Moore, George Varghese, Building a Better Netflow, SIGCOMM, August 2004

[17] Abilene-V Trace Data, http://pma.nlanr.net/Special/ipls5.htm