Time-out Bloom Filter: A New Sampling Method for Recording More Flows

Shijin Kong[1], Tao He[2], Xiaoxin Shao[3], Xing Li[4]

{ksj00[1], sxx03[3]}@mails.tsinghua.edu.cn
Department of Electronic Engineering, Tsinghua University, Beijing, P.R.China, 100084
{hetao[2], xing[4]}@cernet.edu.cn
China Education and Research Network, Beijing, P.R.China, 100084

Abstract. Packet sampling is widely deployed to generate flow records on high speed links. However, random sampling in which 1 in *N* packets is chosen suffers from omitting majority of flows, most of which are short flows (within *N* packets). Although usage-based applications work well by recording long flows and neglecting short ones, there are many other applications which depend on nearly per-flow information other than flow lengths and sizes. In this paper, a novel sampling method is proposed to remedy the flow loss flaw for those applications. We use a Time-out Bloom Filter to alleviate the sampling bias towards long flows. Compared with random sampling, in our solution, short flows have a much greater probability to be sampled while long flows are always sampled, but with much fewer sampled packets. Experimental results show that, with the same sampling rate, our solution records several times more short flows than random sampling. Particularly, up to 99% original flows can be retrieved from sampled data. Besides, we also propose an adaptive TBF system in fast SRAM to perform online sampling.

1 Introduction

With the increasing network traffic, most measurement systems employ packet sampling to reduce resource consumption. First, there is not much resource remained for data collection on routers and switches because of their heavy workload. Forming flow statistics on a sampled substream of the original traffic reduces memory consumption and frequency of flow lookups. Second, transmitting sampled data to collectors, which is common for many applications, can greatly save bandwidth for collection as well as processing and storage cost at collectors. Moreover, some time-consuming processes such as flow records generation can be executed on collectors. The volume of sampled data is small to transmit and the burden on routers and switches is alleviated at the same time.

However, random sampling in which 1 in *N* packets is chosen causes great loss of flow information and it is difficult to recover. A flow is defined as a set of packets with a same key which consists of some fields in packet header. When a packet arrives at the router with a flow key different from those of all current active flows, a new flow record is generated with the new key. A flow is terminated when the time

since the arrival of its latest packet exceeds a time-out threshold. *Flow length* is defined as the number of its packets, and *flow size* is the total bytes in it. If any packet of a flow is sampled, we call this flow is sampled. In random sampling, a short flow (within *N* packets) is easily lost if none of its packets is sampled, and a long flow has a much greater probability to be sampled. The bias towards longer flows causes majority of short flows lost, and thus brings a great loss in total sampled flows since most flows in traffic are short flows (e.g. 82.3% in the trace for our experiments).

Few studies have been addressed to the *flow loss flaw* in random sampling and no sampling method has been designed to meliorate the great loss of flows. Experiments have shown that the distribution of flow lengths is heavy-tailed (see, e.g., [1]), that is, most traffic is carried by a small proportion of long flows. For this reason, sampling methods of many usage-based applications focus on long flows and neglect short ones [13]. However, there are still a lot of applications which depend on nearly per-flow information other than flow lengths and sizes. Here are some examples.

Attacks Detection: information of short flows is very important to detect network intrusions such as port scanning and SYN flooding. These attacks usually consist of numerous short flows with only several packets. It is hard to discover these attacks unless nearly per-flow state is maintained [16], including flow key and correct count of SYN/FIN flags. Moreover, identifying a victim requires enough flow records.

Traffic Identification: in particular, P2P traffic can be effectively identified by using connection patterns [2] instead of checking payloads of packets. This method counts <IP, Port> pairs retrieved from the flow information. Any obvious loss of flow records will cause fallacious counting results and hurt identification accuracy.

Network Deployment Characterizing: the diversity of flow records reflects the spatial distribution of flows in the network. Workloads of network devices (e.g. the size of route tables) can be balanced according to the distribution, which helps to deploy and manage network more efficiently.

In order to meet the needs of those applications, we present a novel packet sampling method in this paper to meliorate the flow loss flaw. Little attention has been paid to exact values of flow lengths and sizes since those applications do not care about them. In our solution, the number of total sampled flows is increased by recording more short flows which are lost in random sampling. Packets are selected by a data structure called Time-out Bloom Filter (TBF). In TBF, some packets can have a great probability to be sampled, and others are definitely discarded. For short flows, considerable proportions of their packets have such a probability. But for long flows, the proportions are very small. Thus, compared with random sampling, our solution has a smaller sampling bias towards long flows. Experimental results show that our method can sample several times more short flows than random sampling with same *sampling rate* (the ratio of sampled packets count to original packets count). Particularly, up to 99% of total original flows can be retrieved from sampled data while random sampling only records 37%.

The rest of this paper is organized as follows. Section 2 reviews related work. Section 3 proposes our measurement method and section 4 makes the comparison with random sampling. Results of measurement on experimental traces are presented in Section 5 and section 6 proposes an adaptive TBF system. Finally, section 7 concludes the whole paper and discusses the future work.

2 Related Work

In this section, we review previous work on flow-related sampling, and *hash-based* applications which are similar to ours.

Classical uniform sampling methods like random sampling are reviewed in Duffield's paper [3]. Flow records on randomly sampled packets are commonly generated by Cisco NetFlow [4] with configurable sampling period N. In [5], an adaptive NetFlow with dynamic sampling rate is devised, and one of its primary contributions is to give accurate numbers of non-TCP flows. Sampling methods for long flows such as smart sampling [6, 7] can give an accurate estimation of total usage for each flow size. Although per-flow detailed information can not be told by these methods, research has been done to estimate the distribution of flow lengths [8, 9].

Hash-based sampling methods have also been applied for several purposes. Trajectory sampling [10, 11] puts particular flow keys in hash tables. Later packets with those keys are selected at each node to detect spatial distribution of flows. Space-Code Filter [12] uses a group of Bloom Filters with different resolutions to estimate flow length of any given key. Multistage Filter [13] selects packets based on several hash functions to identify large flows. And Partial Completion Filters in [14] propose a scalable solution to detect network attacks.

3 Our Sampling Method

In this section, we introduce Time-out Bloom Filter for packets sampling. TBF is derived from standard Bloom Filter [15]. A BF is a hash table with m bits, denoted as $b[0]$, $b[1]$, …, $b[m\text{-}1]$, each of which can be 0 or 1. There are d independent hash functions, $h_1(x)$, $h_2(x)$, …, $h_d(x)$, attached, and each of them maps a given key to one of the m bits. To insert a key c into the table, all the d bits $b[h_1(c)]$, $b[h_2(c)]$, …, $b[h_d(c)]$ are set to 1. Initially there are n keys in the table. Later, the table is used to check whether a given key c' has been inserted. If $b[h_1(c')]$, $b[h_2(c')]$, …, $b[h_d(c')]$ are all set to 1, c' is recognized as one of the n initial keys, otherwise it is not. On average, BF has a complexity of O(1) for querying a given key, which is much faster than traditional hash method with complexity O(m). However, since different keys can have same values calculated by hash functions, each bit can be set by several keys. If all the d bits of a non-initial key have already been set to 1 by initial keys, it will be mistaken as one of them. This mistake is called a *false positive* error.

TBF is used to tell whether an incoming packet can be sampled. Fast query of BF is retained and the false-positive error is reconsidered in TBF. Section 3.1 shows the principles of TBF and in Section 3.2 we explain the motivation for designing TBF.

3.1 Time-out Bloom Filter

TBF is similar to BF except that it does not have m bits, but m buckets instead, each of which contains a timestamp. The m timestamps are denoted as $t[0]$, $t[1]$, …, $t[m\text{-}1]$. Besides, it has a bucket time-out value t_0.

Our algorithm is described as follows. When a new packet with key c and timestamp t comes, the d timestamps, $t[h_1(c)]$, $t[h_2(c)]$, …, $t[h_d(c)]$, are compared with t. If any of the d timestamps, the ith for example, follows $t - t[h_i(c)] > t_0$ (or we say $b[h_i(c)]$ gets time-out), the packet is sampled, otherwise it is discarded. After comparison, all those d timestamps are updated to t even if the packet is not sampled. Figure 1 illustrates this process with d=3.

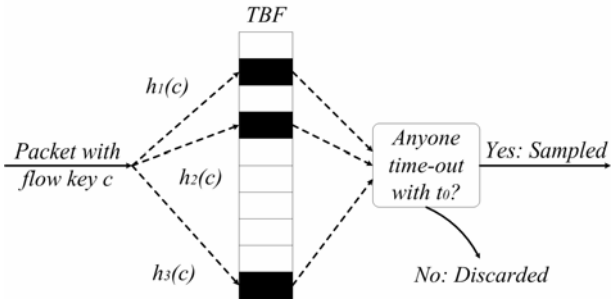

Fig. 1. A Time-out Bloom Filter with d=3

TBF differs from BF in two aspects: **(1)** TBF sets a timestamp for each bucket instead of simply setting it to 1, and updates the timestamps after every packet selection; **(2)** TBF samples a packet as long as any of the d buckets gets time-out while BF requires that all the d bits are set to 1.

3.2 Why Time-out Bloom Filter

Our initial motivation is to sample more flows, so ideally for each flow at least one packet should be sampled. A simple solution is to select the first packet of each flow and discard all the rest packets of it. In this process, we do not have to know any more information of the flow (e.g. IP addresses, ports). The only thing we want to know is whether an incoming packet belongs to an active flow or it is the first packet of a new flow. BF can tell the result quickly. We put all the keys of existed flows in BF and query it every time when a packet comes. According to the query result, an incoming packet is discarded if it is already in BF, or otherwise it is sampled as the first packet of a new flow. Hence a packet is sampled if any of the m bits is not set to 1. This is the origin of the second difference between BF and TBF mentioned in Section 3.1.

Unfortunately, there are two fatal drawbacks using BF.

(1). Not all first packets can be sampled because false positive error happens to mistake the first packet of a new flow for a packet of an existed flow. When this error occurs, the first packet will not be sampled and this flow is lost.

(2). More seriously, the longer the sampling is performed, the more existed flows are put into BF. Finally all m bits will be set to 1. A full-filled BF causes false positive error in every query, rejecting packets of newly generated flows to be sampled. A solution to this drawback is to clear the BF periodically, but the filter will be full-filled quickly on high-speed links (usually less than one second). There is not enough time for any practical sampling implementation to clear BF every second.

Thus TBF is applied to meliorate those drawbacks. In TBF, a bucket getting time-out can be viewed as a bit set to "0" and a bucket not getting time-out as a bit set to "1". When an incoming packet arrives at time t, only flows that have packets updated within $[t\text{-}t_0, t]$ are still kept in the filter. All other buckets are logically set to "0". As time elapsing, the "1" to "0" transforming process is automatically executed. If t_0 is small enough, TBF is never full-filled with "1". Hence drawback **(2)** is avoided.

Each flow can have multiple packets sampled in TBF rather than exactly one. When the time interval between two continuous packets of a flow is smaller than t_0, the latter one is definitely discarded since none of its d buckets gets time-out. When the interval is greater than t_0, the latter one may not be discarded. As long as any of those buckets is not updated by other flows during the interval, the packet is sampled, or otherwise a false positive error occurs. A flow is lost only if all its packets encounter false positive errors, which greatly meliorates drawback **(1)**. But as a result, redundant sampled traffic is generated because flows unnecessarily have multiple packets sampled.

4 Comparing Sampling Methods

In this section, we compare our sampling method based on TBF with random sampling, and then explain why our sampling method can sample more flows.

4.1 Random Sampling: Sampling Bias towards Long Flows

Assuming that 1 in every N packets is selected by random sampling, it is easy to figure out the probability of a packet to be sampled is

$$P_s=1/N \tag{1}$$

Let $F(k)$ denote all the flows with length k and $M(k)$ denote the number of $F(k)$. On average, $k\mathrm{M}(k)/N$ of packets of $F(k)$ are sampled. So the proportion of sampled flows of $F(k)$ has a minimum $1/N$ when $M(k)/N$ of $F(k)$ have one packet sampled for each, and a maximum k/N when k/N of $F(k)$ have all their packets sampled.

Usually, N is greater than 10 so that for k much smaller than N, the proportion of sampled flows is very small. For longer flows with k much greater than N, almost all of them are sampled. The discrepancy of sampling probabilities represents the sampling bias towards long flows.

4.2 TBF: Greater Packet Sampling Probability and Less Biased Sampling

First, we use the conclusion of BF in [15]. The probability that a bucket gets "1" is

$$P_1=1-(1-1/m)^{Ld} \tag{2}$$

where L is the number of $S(t_0)$, the set of flows sampled during previous t_0 interval. L can be measured by placing an empty TBF on the link for t_0 time and counting the flows in it. For example, set m=65,536, d=3, t_0=0.2s, then L for the trace in Section 5

is 5,882 on average (we divide the trace into thousands of t_0 periods, and measure L on each t_0 interval to get the average). Thus, a typical value of P_1 is 0.23. If L does not vary much whenever it is measured, P_1 is always around 0.23.

d	1	2	3	4
P_1'/P_1	0.14/0.09	0.18/0.16	0.27/0.23	0.32/0.30

Table 1. P_1' and P_1 with m=65,536, t_0=0.2s and several d

Now, we suppose a packet with a key c, which does not belong to $S(t_0)$, comes. For each i ($1 \le i \le d$), one would easily expect the probability that $b[h_i(c)]$ gets "1" (denoted as P_1') is P_1. However, that's not the case. P_1' is usually greater than P_1. We think this inconsistency appears due to the correlation among packets. For example, some applications start several flows with same source IP address at nearly the same time. Any hash function only uses source IP address for calculation will map the packets of all these flows to a same bucket. As long as any of these flows is in $S(t_0)$, P_1' for packets of other flows is definitely 1 rather than P_1. Besides, there are many other reasons that cause P_1' to be 1. So on average, P_1' is greater than P_1.

To minimize the correlation, hash functions must be carefully chosen. A hash function should use all fields of flow keys for calculation. Thus, even if two flow keys, c_1 and c_2, are only different in one field, $h_i(c_1)$ and $h_i(c_2)$ are not the same for each i ($1 \le i \le d$). We use such a group of *all-fields-dependent* hash functions in Section 5, and P_1' is measured and compared with P_1 in Table 1. As we see, P_1' is very close to P_1, especially when d >1. The correlation is effectively avoided. Thus, if a packet does not belong to $S(t_0)$, the probability that any of its d buckets gets "0" (time-out) is

$$P_s = 1 - (P'_1)^d \approx 1 - P_1^d \quad (3)$$

P_s is small if d is set to 3 or larger. Again with the example mentioned above, P_1 is 0.23 and P_s is 0.98. It is ***much greater*** than that of random sampling (in Equation 1).

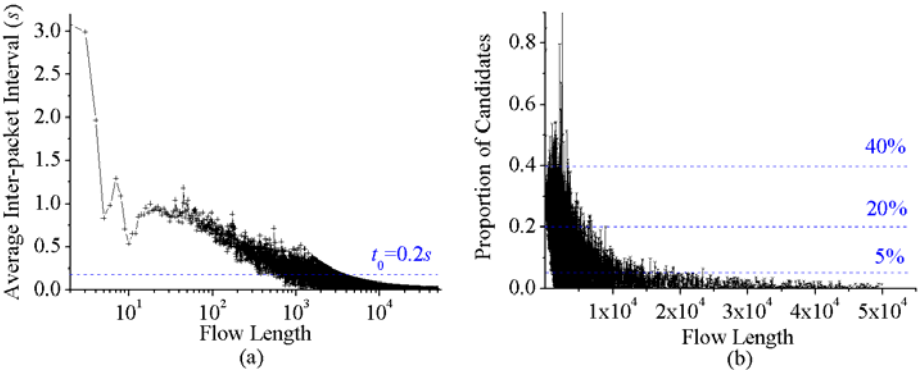

Fig. 2. (a) Average inter-packet interval (b) The proportion of "candidates" for t_0=0.2s

We define a "candidate" as a packet that has a probability P_s to be sampled. A packet that is definitely not sampled is not a "candidate". In random sampling, all the packets are "candidates". In TBF, however, only if the interval between two continuous packets in a flow is greater than t_0, the latter one can be a "candidate". In Figure 2(a), we can see that the longer the flow, the shorter the average inter-packet interval

is. If a proper t_0 is selected, there is small proportions of "candidates" in long flows (<5%) but large proportions of "candidates" in short flows (20% to 40%), as shown in Figure 2(b) with t_0=0.2s. Compared with random sampling, the bias towards long flows is ***reduced*** since fewer packets of long flows are sampled.

In summary, our solution differs from random sampling in two aspects. **(1)** The probability of any packet to be sampled (P_s) is much greater. **(2)** The bias towards longer flows is alleviated because longer flows have fewer proportions of "candidates" than shorter flows. These two aspects determine that TBF can retrieve much more flows from sampled data with same volume (or the same sampling rate) than random sampling.

5 Results of Comparison on Traces

This section shows the experimental results of TBF and its comparison with random sampling. We use a trace containing 681,268,937 packets captured from a gigabytes link, one of the outlets of THUNET (TsingHua University NETwork). It begins at 13:00, Aug 4, 2005 and lasts for an hour. Each packet is recorded with a flow key including four fields in IP header: source IP address, source port, destination IP address and destination port. The time-out value for flow termination is 30 seconds. Totally 8,475,966 flows are generated in the trace.

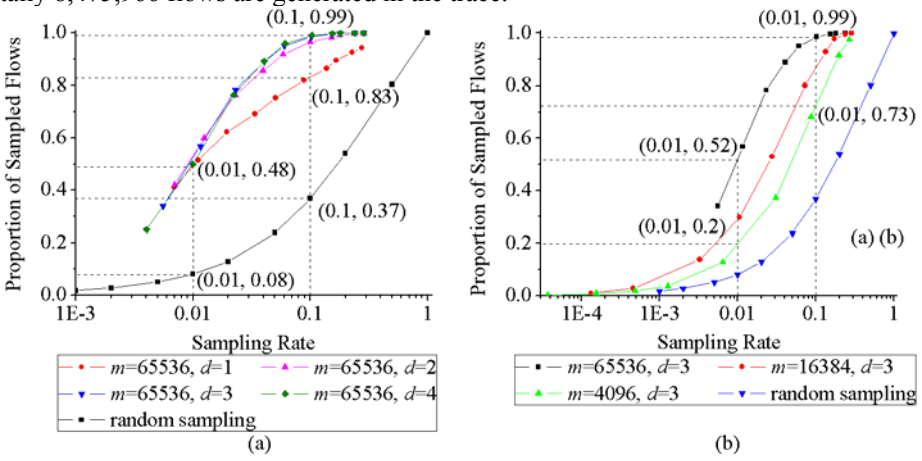

Fig. 3. The proportion of flows sampled by: (a) TBF with fixed m=4 (b) TBF with fixed d=3.

First, we perform random sampling 10 times on the trace with N=1, 2, 5, 10, 20, 50, 100, 200, 500, 1000 respectively. The sampling rate for random sampling is simply 1/N. Then we perform TBF sampling with different m, d and t_0. In our experiments, we simply use different combinations of mask on IP addresses and ports as all-fields-dependent hash functions.

Figure 3(a) shows the proportion of total sampled flows by TBF against the sampling rate with fixed m=65,536 and d=1, 2, 3, 4. Figure 3(b) shows the results with fixed d=3 and m=4,096, 16,384, 65,536. For each combination of m and d, TBF are

performed 10 times with t_0=0.01s, 0.02, 0.05s, 0.1s, 0.2s, 0.5s, 1.0s, 2.0s, 5.0s, 10s respectively. The sampling rate decreases *monotonically* while t_0 increases.

As we can see, TBF samples much more flows than random sampling with same sampling rate. When the sampling rate is 0.1, TBF with m=65,536 and d=3 records 99% of original flows while random sampling (N=10) records 37% of them. The result of TBF is consistent with Equation 3: each flow at least has one candidate (the first packet), so at least P_s=98% of original flows can be sampled. Some flows have multiple candidates, thus results in a higher 99% on the whole. When sampling rate is smaller, say 0.01, random sampling only samples 8% but TBF still records 48%.

Either increasing m or increasing d helps to enhance the number of sampled flows while the sampling rate is kept unchanged. But increasing m is more effective than increasing d. For sampling rate 0.01, varying d from 1 to 4 gives 16% more sampled flows while by changing m from 4,096 to 65,536, 26% extra flows are recorded. We also notice that for d=3 and d=4, the two curves have already overlapped. It means there is no more gain by adding more hash functions.

In Figure 4(a), proportions of sampled flows of $F(k)$ ($1 \le k \le 20$) are shown. For each length k, TBF with m=65,536, d=3 and t_0=0.2s records more than P_s=98% of $F(k)$. The sampling rate of this TBF is about 0.1, so it is compared with random sampling N=10. When k is fewer than five, the number of sampled flows of $F(k)$ recorded by TBF is at least three times more than that by random sampling. To give a clearer view, various flow counts for each k are presented as a percentage of the total original flow count in Figure 4(b). As we expect, the sum of short flow counts, $\Sigma M(k)$ ($1 \le k \le N$=10), represents majority of the total original flow count, 82.3% exactly.

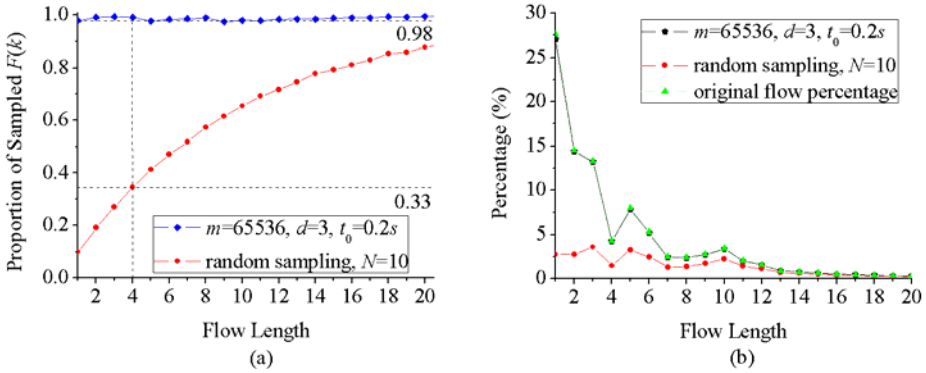

Fig. 4. Sampled flows distributed in flow lengths ranged from 1 to 20, in proportion to (a) $M(k)$ (b) the number of total original flows

6 Adaptive TBF Sampling System

In this section, we focus on implementing adaptive TBF sampling systems. To perform TBF sampling effectively, m, d and t_0 should be carefully chosen.

Although m should be as great as possible, it is limited by memory. In our implementation, we only use a 256KB SRAM for TBF. Since t_0 is at a level of 0.1s, each

bucket records its updating time in milliseconds to keep accuracy. We set m=65,536, and each bucket has 2bytes to store the last 16 bits of packet arriving time in milliseconds. That is $2^6 \times 2^{10}$=64 seconds. To our experience, all buckets are updated within every 64 seconds, so a bucket getting time-out will not be mistaken as NOT-TIMEOUT in the pseudo code of Figure 5.

d is usually set to 3 or 4 as in Section 5 we analyzed that there is no gain to further add hash functions. On the other hand, for every packet, d hash values are calculated. If d is smaller, per-packet process is faster. We set d=3 and continue using the all-fields-dependent hash functions devised in Section 5.

When m and d are determined, t_0 is used to make a tradeoff between the sampling rate and the proportion of flows sampled. If t_0 is small, P_s is great (in Equation 3) and lots of flows are retrieved from redundant sampled data. If t_0 is large, low P_s causes few sampled flows but low sampling rate. In real applications there always exists a target for performance, say, recording more than 90% flows or sampling no more than 10% packets, which helps to choose a proper t_0.

In our implementation, we set the target as sampling no more than 10% packets. The network traffic varies all the time, and a fixed group of m, d, t_0 will cause the sampling rate much higher or lower than 10% sometimes. So we devised an adaptive TBF sampling system to keep sampling rate around the target. In Figure 5, t_0 is adjusted based on the sampling rate measured every five seconds. If the sampling rate is over 0.1, t_0 is set larger and if the sampling rate keeps below 0.1 for 3 continuous measurements, t_0 is set smaller.

```
CHECK_BUCKET_TIMEOUT ( timestamp t, bucket i)
        if ( t > t[i] AND t – t[i] < t0 OR t < t[i] AND t + 64,000–t[i] < t0)
                return NOT-TIMEOUT;
        endif
        return TIMEOUT;
ADAPTIVE_TBF_SAMPLING
        if (sampling_rate > target OR sampling_rate < target for 3 measurements)
                t0 = t0 * (sampling_rate / target)^(1/2);
        endif
```

Fig. 5. Pseudo codes for time-out checking and adaptive TBF sampling

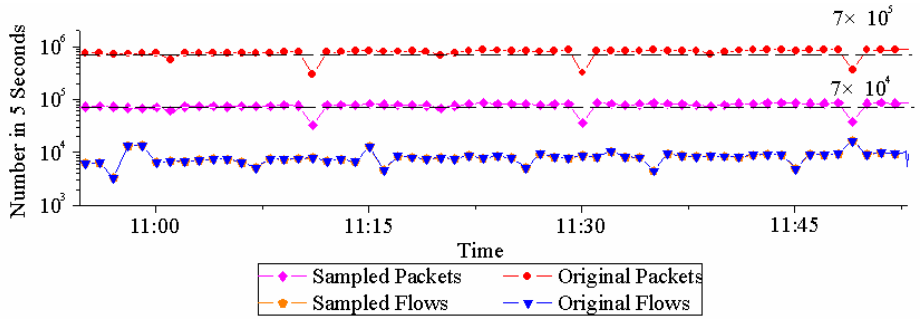

Fig. 6. One-hour result of adaptive TBF sampling system

We already have a gigabytes measurement system based on Intel IXP2400 network processor to capture packet headers from one of THUNET outlets. To test the system

for online sampling, we implement it on a host which receives the captured packet headers from the measurement system. In Figure 6, a one-hour result of the system is shown. On the whole, about 99.3% flows are sampled with sampling rate 0.1.

7 Conclusion and Future Work

In this paper, we propose a novel sampling method based on Time-out Bloom Filter to remedy the short flow loss flaw in random sampling. We analyze the motivation of using TBF for packets selection and the reasons for its smaller sampling bias towards long flows. By comparing with random sampling on network trace, we find that TBF sampling can record up to 99% of total original flows and several times more short flows than random sampling. We have also discussed the proper choices of parameters and then devised an adaptive TBF sampling system for online sampling. In the future work, we are going to program the algorithm on IXP2400 network processor to integrate TBF sampling with the measurement system.

References

1. A. Feldmann, J. Rexford, and R. Cáceres. Efficient Policies for Carrying Web Traffic over Flow-switched Networks. IEEE/ACM Transactions on Networking, 6(6): 673 - 685, 1999.
2. T. Karagiannis, A. Broido, M. Faloutsos, et al. Transport Layer Identification of P2P Traffic. ACM SIGCOMM Internet Measurement Conference (IMC), 2004.
3. N.G. Duffield. Sampling for Passive Internet Measurement: A Review. Statistical Science, 19(3):472 - 498, 2004.
4. Ciso NetFlow. http://www.cisco.com/warp/public/732/netflow/index.html.
5. C. Estan, K. Keys, D. Moore, et al. Building a Better NetFlow, ACM SIGCOMM, 2004.
6. N.G. Duffield, C. Lund, and M. Thorup. Charging from Sampled Network Usage. ACM SIGCOMM Internet Measurement Workshop (IMW), 2001.
7. N.G. Duffield and C. Lund. Predicting Resource Usage and Estimation Accuracy in an IP Flow Measurement Collection Infrastructure. ACM SIGCOMM IMC, 2003.
8. N.G. Duffield, C. Lund, and M. Thorup. Estimating Flow Distributions from Sampled Flow Statistics. ACM SIGCOMM, 2003.
9. N. Hohn and D. Veitch. Inverting Sampled Traffic. ACM SIGCOMM IMC, 2003.
10. N.G. Duffield and M. Grossglauser. Trajectory Sampling for Direct Traffic Observation. IEEE/ACM Transactions on Networking, 9(3):280 - 292, 2001.
11. N.G. Duffield and M. Grossglauser. Trajectory Engine: A Backend for Trajectory Sampling. IEEE Network Operations and Management Symposium, 2002.
12. A. Kumar, J. Xu, J. Wang, et al. Space-Code Bloom Filter for Efficient Per-Flow Traffic Measurement. IEEE INFOCOM, 2004.
13. C. Estan and G. Varghese. New Directions in Traffic Measurement and Accounting. ACM SIGCOMM, 2002.
14. R.R. Kompella, S. Singh, and G. Varghese. On Scalable Attack Detection in the Network. ACM SIGCOMM Internet Measurement Conference, 2004.
15. B.H. Bloom. Space/time Tradeoffs in Hash Coding with Allowable Errors, ACM Communicationsm 13(7), 1970.
16. K. Levchenko, R. Paturi, and G. Varghese. On the Difficulty of Scalably Detecting Network Attacks. ACM CCS, 2004.