Stream Mining for Network Management

IEICE TRANS. COMMUN., VOL.E89–B, NO.6 JUNE 2006

PAPER

Stream Mining for Network Management

Kenichi YOSHIDA[†a)], **Satoshi KATSUNO**[††], **Shigehiro ANO**[††], **Katsuyuki YAMAZAKI**[††], *and* **Masato TSURU**[†††], *Members*

SUMMARY Network management is an important issue in maintaining the Internet as an important social infrastructure. Finding excessive consumption of network bandwidth caused by P2P mass flows is especially important. Finding Internet viruses is also an important security issue. Although stream mining techniques seem to be promising techniques to find P2P and Internet viruses, vast network flows prevent the simple application of such techniques. A mining technique which works well with extremely limited memory is required. Also it should have a real-time analysis capability. In this paper, we propose a cache based mining method to realize such a technique. By analyzing the characteristics of the proposed method with real Internet backbone flow data, we show the advantages of the proposed method, i.e. less memory consumption while realizing real-time analysis capability. We also show the fact that we can use the proposed method to find mass flow information from Internet backbone flow data.

key words: *stream, mining, network, management*

1. Introduction

Network management is an important issue to maintain the Internet as an important social infrastructure. The treatment of P2P and Internet viruses are the two most important issues of network management. Since the vast consumption of network bandwidth caused by P2P mass flows is becoming excessive, a method to find and prevent them is an important network management task to keep the Internet working optimally. To protect Internet security, finding Internet viruses is also an important issue.

Since both P2P and Internet viruses are rapidly making new varieties, automatic finding methods of their new varieties are desired. Stream mining techniques seem to be promising techniques to find new P2P and Internet viruses automatically. However, vast network flows prevent the simple application of such techniques. A stream mining technique which works well with extremely limited memory is required. Also it should have a real-time analysis capability. For example, 10 Gbps of network bandwidth can transfer 100 Tera bytes of data per day. Since today's Internet backbone has a broader bandwidth, a mining system has to handle more than 100 Tera bytes of data per day. Although a large computer with Giga bytes of memory can be used, the memory size of such a computer is still extremely small if we compare it to the amount of data to be analyzed. The real-time analysis capability is also indispensable.

In this paper, we propose a cache based mining algorithm. The original concept of the proposed algorithm is the use of a fixed size cache memory to find frequent items. Through the best use of the fixed size cache, we hope to realize a stream mining method which can work well with extremely limited memory resources while realizing real-time analysis capability.

By analyzing the characteristics of the proposed method with real Internet backbone flow data, we show the advantages of the proposed method. We also show the fact that we can use the proposed method to find mass flow information from the Internet backbone flow data.

Section 2 of this paper first surveys related work and determines their limitations in order to clarify the motivation of this research. Section 3 explains our methods, and Sect. 4 reports on the experimental results. Section 5 examines characteristics of the proposed method. Finally, Sect. 6 concludes our findings.

2. Related Work

Monitoring Internet traffic is an extensively studied area, e.g. [1]–[4]. IETF's IPPM working group proposes a framework of IP performance metrics [3]. Their work is important in providing a baseline to compare the measured results by standardizing the attributes to be measured. Surveyor [4] is a project to create measurement infrastructure. NLANR [2] has a project to develop a large-scale data collection system as the base infrastructure for various data analysis. CAIDA [1] is making various tools to analyze network data. Their visualization tools cover various analysis of network data.

Analysis of measured data is also studied [5], [6]. Some studies, e.g. [7], try to use data mining techniques to automate analysis. Though [7] claims its functionality with restricted memory, further research is necessary.

When considering the importance of data mining performance, various methods for frequent item finding have been proposed. Among them, Coarse counting [8], Sticky sampling, Lossy Counting [9], hash-based approaches [10], [11], and the use of group test [12] are important methods. These methods can quantify frequently appearing items

Manuscript received September 28, 2005.
Manuscript revised February 9, 2006.

[†]The author is with the Graduate School of Business Science, University of Tsukuba, Tokyo, 112-0012 Japan.
[††]The authors are with KDDI R&D Laboratories Inc., Fujimino-shi, 356-8502 Japan.
[†††]The author is with the Department of Computer Science and Electronics, Faculty of Computer Science and Systems Engineering, Kyushu Institute of Technology, Iizuka-shi, 820-8502 Japan.
a) E-mail: yoshida@gssm.otsuka.tsukuba.ac.jp

DOI: 10.1093/ietcom/e89–b.6.1774

without any omissions. However, we found that these methods had poor performance when working with limited memory. These methods tend to overestimate the frequencies of occurrence when the available memory is limited.

In this study, we investigate a method which uses a fixed size cache memory to find frequent items. The management of cache memory significantly affects the performance of our method. The study of cache management has a long history. LRU based methods such as LFU, LRU-k [13], and 2Q [14] form an important family of management strategies. However, extensive study on the use of these methods for frequent item finding has not been reported.

The memory management strategies we used in this study, i.e. random2 and hash2, retain information on multiply accessed entries. Random2 was originally developed in the study of spam filters [15] and its characteristics were reported in [16]. In this paper, we introduce hash2 as an enhancement of random2.

Among conventional studies, CPM [16] and Space-Saving [17] has the best performance with restricted memory. We empirically show the advantage of our new method in Sect. 4. Comparison with the second best method, i.e. hCount∗ [11], is also reported in this paper. Note that most of the mining methods which work well with limited memory, e.g. CPM, Space-Saving and hCount∗, are off line methods. They tend to lack the function of handling so called concept drift [18]. The method we proposed in this paper has the ability to handle concept drift while retaining performance under limited memory.

Finally, from the view point of flow measurement, there exist a considerable number of works to measure huge traffic on high-speed links such as core routers in the Internet backbone. Note that a flow here is a sequence of packets having the same five-tuples, that is, source/destination addresses, source/destination ports, and protocol. To measure (process) a huge number of packets per unit time on a very high-speed link, packet sampling is often employed such as NetFlow and sFlow in commodity routers. For retrieving the original flow information from the sampled data, several methods have been proposed for obtaining flow statistics [19], [20] and for counting the frequency (i.e., the number of packets) of each of flows frequently appeared, which is often called an elephant flow [21]. On the other hand, to cope with the memory limitation of measurement systems to record the per-flow information on a huge numbers of flows, the use of a kind of irreversible compression of information by Bloom filter or its extension (space-code Bloom filter) has also been proposed for counting only elephant flows [22] or for roughly counting all flows in a multi-resolution way [23]. However, these all methods generally suffer from overhead of complex off-line processing and difficulty of finding appropriate parameters of them for archiving a reasonable accuracy of counting, due to the nature of statistical inference from sampled and/or compressed information.

In contrast with them, our method proposed in this paper is very light-weight and memory-efficient because it just counts the frequency of each of appeared flows in a fixed-sized table with a novel table-entry replacement strategy, while it can find and count elephant flows in an on-line manner with a reasonable accuracy. Note that, a sliding window-based on-line method of counting elephant flows has been proposed [24], but it is still complex compared with our proposed method. In general, a window-based method requires both auxiliary buffer to store the item information in the sliding window and periodic operations over the sliding window. It suffers from the trade-off between accurate but costly fine-grained sliding and inaccurate but light coarse-grained sliding. In the algorithm proposed in [24], the accuracy, memory requirement, and computational overhead for each step are sensitive to the sliding window size. In addition, the existence of appropriate window size is not proven for a huge scale data (in [24], total number of distinct flows is just 1647).

3. CPM-Stream & Hash2

We have investigated an off line version of CPM (Cache-based Pattern Mining) which uses a fixed size cache memory to find frequent items [16]. Figure 1 shows its algorithm. It simply counts the frequency of items. A fixed size cache was used to store frequencies.

If the size of cache is large enough, calculating the index of a cache entry for an item is simple (Fig. 1, 5th line). A standard hash technique can be used to find the index for the recently encountered item[†]. Free entries can be used for the newly encountered items. When memory is restricted, cache entries have to be reused by deleting old entries in the cache memory. How an entry is selected to be deleted significantly affects the performance of CPM. The memory management strategy, Random2, (Fig. 2) is the strategy CPM uses in such cases.

Among conventional studies, CPM and Space-Saving show the best performance when available memory is limited [16], [17]. However, both CPM and Space-Saving are essentially off line algorithms and cannot handle so called

```
Algorithm CPM
begin
    Create empty heap;
    while (input item) do
        i = index of item in heap;
        increment heap_cnt[i] by 1;
        if (heap_cnt[i]>thresh_hold)
            print message;
    done
end
```

Fig. 1 Algorithm of CPM.

```
int i = random() % HEAP;
for (p=1; (heap_cnt[i]>p); p++)
   i = random() % HEAP;
```

Fig. 2 C program code of Random2.

[†]This requires an auxiliary hash table. The need for this hash table is omitted in hash2. See later in this Section.

```
Algorithm CPM-Stream
begin
    Create empty heap;
    while (input Item) do
        Idx = Find Index of Item in Heap;
        increment heap_cnt[Idx] by 1;
        if (heap_cnt[Idx]>thresh_hold)
            print message;
        i = randomly select heap element;
        decrement heap_cnt[i] by 1;
    done
end
```

Fig. 3 Algorithm of CPM-Stream.

concept drift [18]. For example, when a user tries to transfer data using P2P software, a P2P flow starts at some point in time, and ends after it transfers the data of intention. During that period, the packets of this P2P traffic appear frequently. However, after the P2P completes its data transfer, they are not frequent.

Since original CPM cannot process this, we have developed an on-line version. Figure 3 shows the on-line version of CPM, named CPM-Stream, and Fig. 5 shows its memory management strategy hash2.

To handle concept drift, CPM-Stream randomly decreases one counter when its increments another counter for the new item (See under lined line in Fig. 3). By doing this, CPM-Stream handles concept drift. Even if the frequency of some items is large, it gradually becomes small as long as the item does not appear again.

We also enhance the memory management strategy (See Figs. 4 and 5). Hash2 is an enhancement of random2. When the memory management of CPM-Stream do not find entry for the given item, it has to make space by deleting old entry. To do so, it use hash2 to make space for new entry by deleting old entry (See last part of Figs. 4 and 5). It first calculates N hash values of a given item. N hash functions are used for this purpose. Next it generates N indexes from N hash values. Then hash2 selects the index which refers to the least frequent entry out of N entries referred by N indexes. We used 4 as N in the experiments reported in the next section. Although we did not extensively seek the best N, 4 tends to make reasonably good results in various experiments.

The entries in the cache are to be deleted only when the memory management of CPM-Stream does not find the entry for the given new item. When the memory management of CPM-Stream does not find the entry for the given item, hash2 selects an entry to be deleted and deletes it. Note that, CPM-Stream decreases one counter of a randomly selected item (underlined in Fig. 3) every time a packet comes, and decreasing counter of an item leads to increasing the probability that the item will be deleted by hash2. However, the delete operation itself will be performed by hash2.

Although both hash2 and random2 have a mechanism to implement the "Retaining multiply accessed entries" strategy, they select entries to be deleted randomly. Random function is used by random2. Hash2 uses a hash function as a substitute for the random function. Although random2 needs an auxiliary hash table to find the index for the recently encountered item, hash2 does not require such table. Hash2 can find the index for the recently encountered item by calculating its hash values (Fig. 4 8 ∼ 10 lines). Thus, the memory efficiency of hash2 is slightly improved upon from random2.

```
Function Find Index of Item in Heap
Input
    Item: Data to be stored in Cache
Variable
    Hash[]: Table of Hash Values
    Idx[]: Table of Cache Index
begin
    Calculate N hash values from Item
            and store them into Hash[]
    Idx[] = Hash[] % Cache Size
    if (one of entry refereed by Idx[] stores Item)
    then  return Idx that refers the entry
    else  Calculate Idx by Hash2
          heap_cnt[Idx] = 0
          return Idx
end
```

Fig. 4 Pseudo code of memory management strategy.

```
Function Hash2
Input
    Item: Data to be stored in Cache
Variable
    Hash[]: Table of Hash Values
    Idx[]: Table of Cache Index
begin
    Calculate N hash values from Item
            and store them into Hash[]
    Idx[] = Hash[] % Cache Size
    return Idx that refers least frequent entry
end
```

Fig. 5 Pseudo code of Hash2.

4. Experimental Results

In this section, we analyze an IP header log with CPM-Stream and other methods. The IP header log was recorded on a monitoring point of a commercial Internet backbone, and is a collection of MD5 values containing 164 million IP packets. Only the source IP address, destination IP address, source port number, and destination port number are used to calculate MD5 values. Since MD5 values and original data, i.e., set of IP address and port number, has one-to-one correspondence, finding frequently appearing MD5 values from this data means finding frequent/mass flows caused by P2P traffic[†].

Because of the file system limitation of the operating system we used, the IP header log is stored in multiple files. The size of each file is 2G bytes. The first file stores the packets of first period. The second file stores the succeeding period, and so on. About 900 sub-files are used to store the entire IP header file.

[†]The use of MD5 value is to make experimentations easy. By using MD5 value, auxiliary programs which calculate basic statistics become simpler. The size of data file also becomes smaller. In the real situation, the use of MD5 is not necessary. Here we just select MD5 to make experimentations easy in a way which does not affect the accuracy of the results.

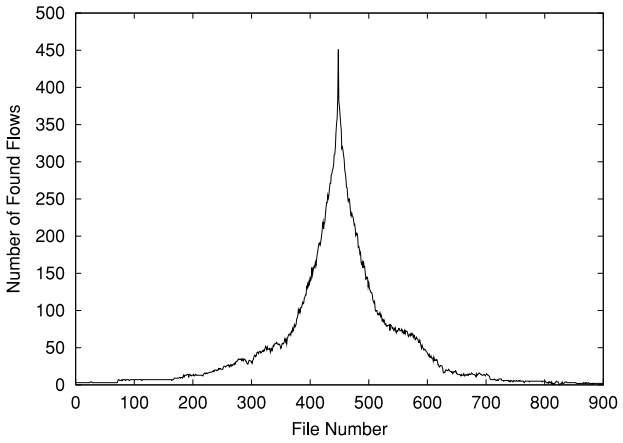


Fig. 6 Frequency drift of IP flows.

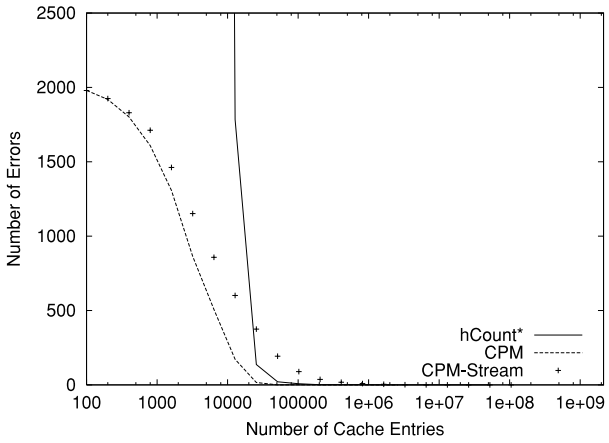


Fig. 7 Comparison of memory efficiency.

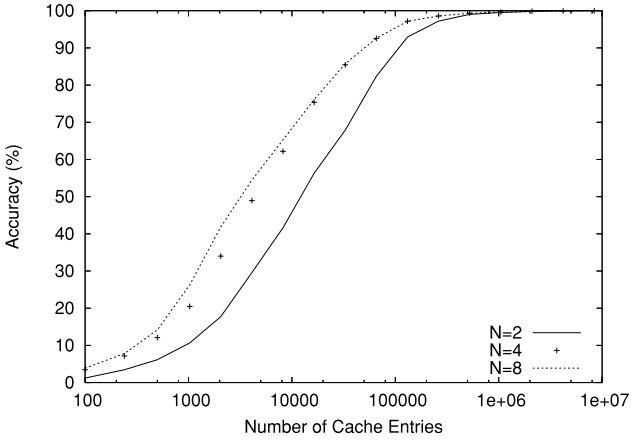


Fig. 8 Effect of cache size and number of cache function.

4.1 Concept Drift in Internet Data

We first check how the frequency of flow changes. To do this, we make a list of MD5 values which appear more than 1,000 times in the 450th file. Next we check how many of them also appear in other files. Here a PC with sufficient memory to hold all the information was used. Figure 6 shows the results. The vertical axis shows how many of the MD5 values that appear on the 450th file also appear in the other file. The horizontal axis is the file number and represents the time sequence.

As clearly shown in the figure, the frequency of flows are changing. As the file numbers' difference becomes large, i.e. as the difference of the files' creation time becomes large, the number of found flows in both files becomes small. This is due to the drift of flow frequencies. Thus stream mining methods have to handle concept drift to analyze this drift of flow frequencies.

Note that in each of the files which store IP header information, packets of the top flows which have more than 1,000 packets represent about 50% of all of the packets. Thus finding these top flows can contribute to finding mass flows.

4.2 Memory Efficiency

As we explained before, performance under limited memory resources is important in applying mining methods to analyze network data. Among conventional studies, original CPM [16] has the best performance under limited memory conditions. Recently [17] reports Space-Saving which has similar memory performance with theoretical upper bound of memory usage. hCount$*$ [11] has the second best performance. Figure 7 compares CPM-Stream, original CPM (i.e. off line version of CPM), and hCount$*$.

In this experiment, we measured the performance of each method by changing the memory size, i.e. the number of cache entries. The entire IP header log data (i.e. 164M packets data of 2.68Mflows) was used. The flows whose packets appear more than 10,000 times are marked as frequent. 2050 flows meet this condition in the data and 73% of packets belong to these flows.

CPM-Stream and CPM make underestimation errors. They both underestimate the frequency of data under limited memory conditions and miss frequent items. hCount$*$ makes overestimation errors. It overestimates frequency of data under limited memory conditions and misidentified non-frequent items as frequent. Figure 7 compares the number of errors generated by these methods.

As shown in the figure, CPM is always best and CPM-Stream is next. hCount$*$ is the worst among these 3 methods. The performance degradation of CPM-Stream from CPM is due to the handling of concept drift. The random decrease of the counter (See under lined line in Fig. 3) causes the performance degradation. However, as described in next sub-section, this performance degradation can be compensated when handling of concept drift is necessary. For example, Fig. 7 shows an area where hCount$*$ outperforms CPM-Stream. However as we describe in the next session, we believe the benefit of concept drift handling is more important than this performance degradations.

Figure 8 shows the effect of memory size and number of hash functions used in hash2. In this figure, horizontal axis is the number of cache entries, and vertical axis is the

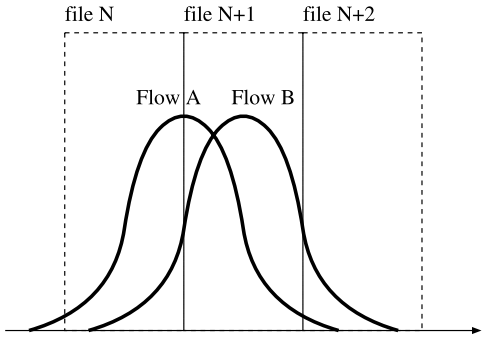


Fig. 9 Effect of online analysis.

percentage of found frequent flows. 100% means that CPM-Stream finds all 2050 frequent flows. As shown in the figure, we can improve the performance of CPM-Stream by increasing the number of hash functions used in hash2. However, larger N also requires larger processing cost. We can also improve the performance of CPM-Stream by increasing the memory size. The performance (i.e., accuracy), processing cost, memory resources are in trade-off situation. Since the memory for 1M entries is small enough in the practical situation and the use of 4 hash functions is enough with 1M entries to achieve good accuracy, we use 4 hash function in the following experiments[†].

4.3 Effect of Concept Drift

Since we store the IP header log in multiple files, we can find mass flows by analyzing each file using an off line data mining program such as CPM and hCount∗. Figure 9 shows a problem of such off line analysis. Since the occurrence of some flows are stored in multiple files separately, off line analysis underestimates the frequency of such flows. For example, off line analysis underestimates the frequency of flow A in Fig. 9. Off line analysis can only estimate the frequency of flow B in this case.

Figures 10, 11, and 12 show the results made by CPM-Stream using different cache sizes (6400, 25600, and 102400 entries respectively). Each figure shows the number of found flows with CPM-Stream and the difference of the results between CPM-Stream and off line CPM. Here CPM used enough memory to find all frequent flows. CPM-Stream continuously inputs multiple files in time sequence. Thus the flows only found by off line CPM is due to the insufficient memory and the flows only found by on line CPM-Stream is due to the phenomenon shown in Fig. 9.

As shown in those figures, the number of flows only found by off line CPM decreases as the memory size of CPM-Stream increases. And the number of flows only found by CPM-Stream increases as the memory size of CPM-Stream increases. With 25,600 cache entries, the number of flows only found by off line CPM and the number of flows only found by CPM-Stream are roughly equal. With cache entries more than 25,600, the number of found flows by CPM-Stream outperforms that by off line CPM due to the proper handling of concept drift.

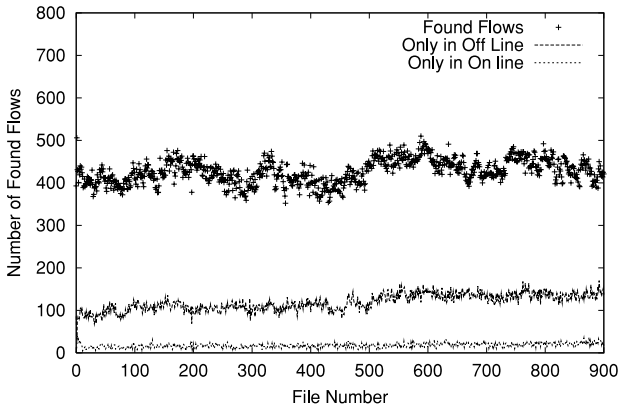


Fig. 10 With 6400 cache entries.

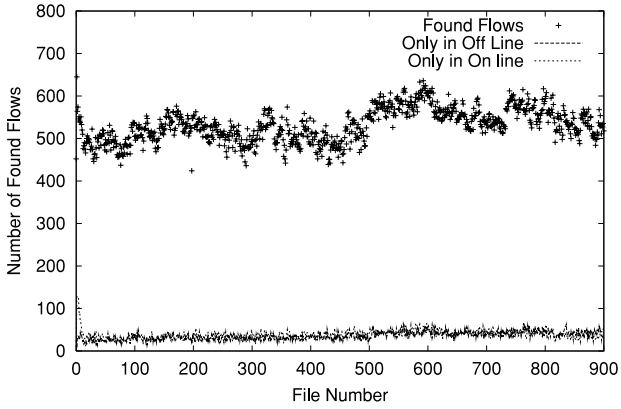


Fig. 11 With 25600 cache entries.

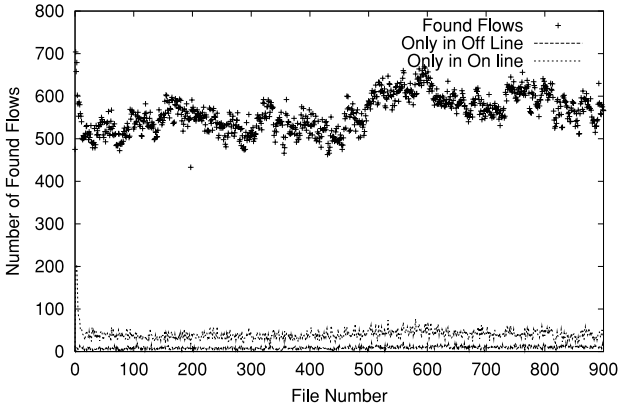


Fig. 12 With 102400 cache entries.

As mentioned in Sect. 4.2, the handling of concept drift slightly decreases the memory efficiency of CPM-Stream from the off line version of CPM. However, if the target data requires the analysis of concept drift, the proper handling of concept drift compensates the memory efficiency degrada-

[†]We also checked the effect of hash program in the auxiliary experimentations. Since the change of hash program has negligible effects on the results and the results by the program in the standard C library were good enough, we only show the experimental results where we change the number of hash functions used in hash2.

tion.

5. Discussion

CPM-Stream is a modified version of CPM which is originally suitable for off line analysis. Two important modifications are the handling of concept drift and the replacement of the memory management strategy from random2 to hash2. The importance of concept drift handling is examined in the previous section.

Figure 13 shows how random2 and hash2 select entries to be deleted. Both memory management strategies are using the retaining multiply accessed entries strategy. They both select less frequently accessed entries as the candidate for deletion. To realize this selection, random2 increases a counter ("p" in Fig. 2) in the loop. Here the counter acts as the threshold of frequency. When the heap size is small, the frequency of the entries in the heap tend to be high. By gradually increases the counter, random2 tries to select less frequent entries in the heap. Hash2 selects the least frequent entry out from randomly selected entries. This statistically selects less frequent entries in the heap.

Another important characteristic of random2 and hash2 is that they both retain newly encountered data for a certain period. When CPM and CPM-Stream encounter new data, the frequency counter of the new data is set to be one. When the counter is small, they are entries to be deleted if random2 and hash2 find them as the candidates for deletion. However, since both random2 and hash2 randomly select candidates for deletion, the entry for the newly encountered data gets a postponement. If the newly encountered data is frequent, the counter of the data will increase rapidly. Thus random2 and hash2 will not delete them and CPM and CPM-Stream can find new frequent data.

We believe that we can realize a different implementation of the "Retaining multiply accessed entries" strategy. Random2 and hash2 are the first attempts. However, the two characteristics discussed above are important to design new implementations of the "Retaining multiply accessed entries" strategy.

Another important issue related to the memory management is how to confirm the sufficiency of the memory size used by CPM-Stream. CPM-Stream does not guarantee the quality of its analysis. If the memory size that the CPM-Stream used was too small, the results miss many frequent items. However, we can check its quality by comparing the result of CPM-Stream with that of the off line version CPM as shown in Figs. 10, 11, and 12. Note that the off line analysis of CPM has to analyze the data which is monitored in some time period. Since the data monitored in some time period is far smaller than the total data, we can use enough memory to find all the frequent items. Thus we can check the quality of CPM-Stream results by comparing it with the results of the off line analysis.

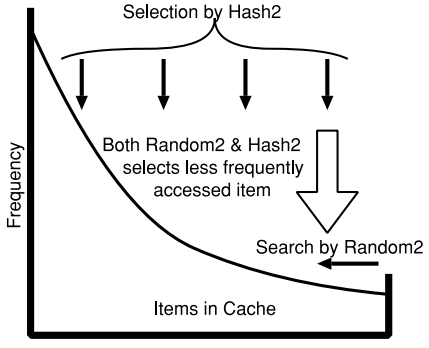


Fig. 13 Selection of Random2 & Hash2.

Concept drift handling of CPM-Stream rely on its memory management strategy hash2. For example, after the file transfer of P2P terminates its massive data transfer, the counter of its flow starts decreasing. Since main routine of CPM-Stream randomly decreases the counter in the cache (See under lined line in Fig. 3), the average decrement of the counter is $\frac{1}{\text{Cache Size}}$ per each new packet. Thus, the counter of the terminated massive flow becomes zero after CPM-Stream process "Number of packets of terminated flow"*"Cache Size" new packet.

Hash2 may delete this entry before the counter becomes zero. The user may start the same flow before the counter becomes zero. Thus the behavior of CPM-Stream is more complicated. However, based on the experimental results shown in the previous section, we find the behavior of CPM-Stream is quite natural. It tends to delete old terminated flow information naturally. It may increase the counter of old discontinued flow's counter. However, counting such discontinued flow as frequent flow does not cause any practical problem.

6. Conclusion

Network management is an important issue to maintain the Internet as an important social infrastructure. Finding excessive consumption of network bandwidth caused by P2P mass flow is especially important. In this paper, we show how we can use a stream mining technique to analyze P2P mass flow with experimental results. The characteristics of our approach are:

- A mining technique which works well with extremely limited memory
- The handling of concept drift to capture the current mass flow
- The "Retaining multiply accessed entries" strategy to best utilize the available memory resources

The effect of the handling of concept drift and behavior of the "Retaining multiply accessed entries" strategy, i.e. random2 and hash2, are also discussed with experiments which use actual Internet traffic. The experiments also show that we can use the proposed approach to find mass flow from Internet backbone flow data.

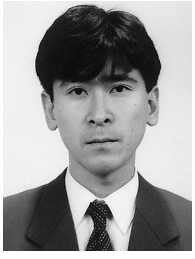
References

[1] http://www.caida.org/

[2] http://moat.nlanr.net/
[3] Rfc2330, framework for ip performance metrics.
[4] http://www.advanced.org/surveyor/
[5] J. Mirkovic, G. Prier, and P.L. Reiher, "Attacking ddos at the source," Proc. 10th IEEE International Conference on Network Protocols, pp.312–321, 2002.
[6] Y. Ohsita, S. Ata, M. Murata, and T. Murase, "Detecting distributed denial-of-service attacks by analyzing tcp syn packets statistically," Proc. IEEE Globecom 2004, vol.4, pp.2043–2049, 2004.
[7] E.D. Demaine, A. Lopez-Ortiz, and J.I. Munro, "Frequency estimation of internet packet streams with limited space," Proc. 10th Annual European Symposium on Algorithms, 2002.
[8] M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, and J.D. Ullman, "Computing iceberg queries efficiently," Proc. 24th Int. Conf. Very Large Data Bases, VLDB, pp.299–310, 1998.
[9] G. Manku and R. Motwani, "Approximate frequency counts over data streams," Proc. 28th International Conference on Very Large Data Bases, pp.346–357, Hong Kong, China, 2002.
[10] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," Proc. Int. Colloquium on Automata Language and Programming, pp.693–703, 2002.
[11] C. Jin, W. Qian, C. Sha, J.X. Yu, and A. Zhou, "Dynamically maintaining frequent items over a data stream," Proc. twelfth international conference on Information and knowledge management, pp.287–294, 2003.
[12] G. Cormode and S. Muthukrishnan, "What's hot and what's not: Tracking frequent items dynamically," Proc. Principles of Database Systems, pp.296–306, 2003.
[13] E.J. O'Neil, P.E. O'Neil, and G. Weikum, "The LRU-K page replacement algorithm for database disk buffering," Proc. ACM SIGMOD International Conference on Management of Data, pp.297–306, 1993.
[14] T. Johnson and D. Shasha, "2Q: A low overhead high performance buffer management replacement algorithm," Proc. Twentieth International Conference on Very Large Databases, pp.439–450, Santiago, Chile, 1994.
[15] K. Yoshida, F. Adachi, T. Washio, H. Motoda, T. Homma, A. Nakashima, H. Fujikawa, and K. Yamazaki, "Density-based spam detector," KDD2004, pp.486–493, 2004.
[16] K. Yoshida, S. Katsuno, Y. Fujita, M. Tsuru, S. Ano, and K. Yamazaki, "Lru is not better than random!," IEICE Trans. Commun. (Japanese Edition), vol.J88-B, no.10, pp.2012–2021, Oct. 2005.
[17] A. Metwally, D. Agrawal, and A.E. Abbadi, "Efficient computation of frequent and top-k elements in data streams," ICDT, ed. T. Eiter and L. Libkin, Lecture Notes in Computer Science, vol.3363, pp.398–412, Springer, 2005.
[18] G. Widmer and M. Kubat, "Learning in the presence of concept drift and hidden contexts," Mach. Learn., vol.23, no.1, pp.69–101, 1996.
[19] N. Duffield, C. Lund, and M. Thorup, "Estimating flow distributions from sampled flow statistics," Proc. ACM SIGCOMM, pp.325–336, Karlsruhe, Germany, Aug. 2003.
[20] N. Hohn and D. Veitch, "Inverting sampled traffic," Proc. ACM SIGCOMM Internet Measurement Conference, pp.222–233, Miami, USA, Oct. 2003.
[21] T. Mori, M. Uchida, R. Kawahara, J. Pan, and S. Goto, "Identifying elephant flows through periodically sampled packets," Proc. ACM SIGCOMM Internet Measurement Conference, pp.115–120, Taormina, Sicily, Italy, Oct. 2004.
[22] C. Estan and G. Varghese, "New directions in traffic measurement and accounting," Proc. ACM SIGCOMM, pp.323–336, Pittsburg, Aug. 2002.
[23] A. Kumar, J. Xu, J. Wang, O. Spatscheck, and L. Li, "Space-code bloom filter for efficient per-flow traffic measurement," Proc. IEEE Infocom, Hong Kong, March 2004.
[24] L. Golab, D. DeHaan, E. Demaine, A. Lopez-Ortiz, and J.I. Munro, "Identifying frequent items in sliding windows over on-line packet streams," Proc. ACM SIGCOMM Internet Measurement Conference, pp.173–178, Miami, USA, Oct. 2003.

Kenichi Yoshida received his Ph.D. from Osaka University in 1992. In 1980, he joined Hitachi Ltd., and is working for University of Tsukuba from 2002. His current research interest includes application of internet and application of machine learning techniques.

Satoshi Katsuno received the B.S. degree and the M.S. degree in electric engineering from the University of Tokyo, Japan in 1989 and 1991, respectively. He joined Kokusai Denshin Denwa Co., Ltd. (KDD), in 1991. He worked at the Tokyo Research & Operation Center of Telecommunication Advancement Organization (TAO) from 2001 to 2004. Since April 2004, he has been at KDDI R&D Laboratories Inc. He has been engaged in research on network quality measurement.



Shigehiro Ano received the B.E. and the M.E. degrees in electronics and communication engineering from Waseda University, Japan in 1987 and 1989, respectively. Since joining KDD in 1989, he has been engaged in the field of ATM switching system and ATM networking. His current research interests are network management over the next generation Internet, QoS routing architecture and multicast protocol for IP broadcasting. He is currently the Senior Manager of IP Communication Quality Lab. in KDDI R&D Laboratories Inc. He received IPSJ Convention Award in 1995.

Katsuyuki Yamazaki received B.E. and D.E. degrees from the University of Electro-communications and Kyushu Institute of Technology in 1980 and 2001, respectively. At KDD Co. Ltd., he had been engaged in development of ISDN and S.S. No.7, R&D and international standards of ATM networks, consultation for a new telecommunication company, and R&D of Internet QoS and networking. He is currently at KDDI R&D Labs. Inc., and responsible for R&D strategy.

Masato Tsuru received B.E. and M.E. degrees from Kyoto University, Japan in 1983 and 1985, respectively, and then received his D.E. degree from Kyushu Institute of Technology, Japan in 2002. He worked at Oki Electric Industry Co., Ltd., Nagasaki University, Japan Telecom Information Service Co., Ltd., and Telecommunications Advancement Organization of Japan. Since April 2003, he has been an Associate Professor in the Department of Computer Science and Electronics, Kyushu Institute of Technology. His research interests include performance measurement, modeling and analysis of computer communication networks. He is a member of the IPSJ, JSSST, and ACM.