Bounds for Frequency Estimation of Packet Streams*

P. BOSE
Carleton University, Canada

E. KRANAKIS
Carleton University, Canada

P. MORIN
Carleton University, Canada

Y. TANG
Carleton University, Canada

Abstract

We consider the problem of approximating the frequency of frequently occurring elements in a stream of length n using only a memory of size $m \ll n$. This models the process of gathering statistics on Internet packet streaming using a memory that is small relative to the number of classes (*e.g.* IP addresses) of packets. We show that when some data item a occurs αn times in a stream of length n, the FREQUENT algorithm of Demaine *et al.* [4], can approximate a's frequency with an error of no more than $(1-\alpha)n/m$. We also give a lower-bound of $(1-\alpha)n/(m+1)$ on the accuracy of any deterministic packet counting algorithm, which implies the FREQUENT algorithm is nearly optimal. Finally, we show that randomized algorithms can not be significantly more accurate since there is a lower bound of $(1-\alpha)\Omega(n/m)$ on the expected accuracy of any randomized packet counting algorithm.

1 Introduction

We consider the problem of processing a data stream $x_1, \ldots, x_n$ of *packet classes* in one pass. This models the process of gathering statistics on Internet packet streams using a memory that is small relative to the number of classes or categories of packets.

*This work was partly funded by NSERC (Natural Sciences and Engineering Research Council of Canada) and MITACS (Mathematics of Information Technology and Complex Systems) grants.

More formally, we consider *packet counting algorithms* that process the stream $x_1, \ldots, x_n$ one item at a time. A packet counting algorithm has a memory of fixed-size and has access to m integer counters, each of which can be labelled with a packet class. If a counter is labelled with some packet class a then we say that counter is *monitoring* a. While processing an item, the algorithm may modify its memory, perform equality tests on packet classes, increment or decrement counters and change the labels of counters. However, other than comparing packet classes and storing them as counter labels, the algorithm may not do any other computations on storage of packet classes. After the algorithm completes, the *counter value* for a packet class a is the value of the counter monitoring a. If no counter is monitoring a then the counter value for a is defined to be zero.

The problem of accurately maintaining frequency statistics in a data stream has applications in Internet routers and gateways, which must handle continuous streams of data that are much too large to store and postprocess later. As an example, to implement fairness policies one might like to ensure that no user (IP address) of a router or gateway uses more than 1% of the total available bandwidth. Keeping track of the individual usage statistics would require (at least) one counter per user and there may be tens of thousands of users. However, the results in this paper imply that, using only 99 counters, we can identify a set of users, all of whom are using more than 1% of the available bandwidth and which contains every user using more than 2% of the bandwidth. If more accuracy is required, we could use 990 counters, and the threshold values become 1% and 1.1%, respectively.

1.1 Related Work

Motivated mainly by applications like the one described above, there is a growing body of literature on algorithms for processing data streams [1, 2, 3, 5, 6, 8, 9, 10, 11, 13]. An early work, particularly relevant to the current paper, is the work of Fischer and Salzberg [7] who showed that, using one counter and making one pass through a data stream, it is possible to determine a class a such that, if any element occurs more than $n/2$ times, then it is a.

Demaine *et al.* [4] showed that Fischer and Salzberg's algorithm generalizes to an algorithm which they call FREQUENT. The same algorithm was also independently discovered by Karp *et al.* [12]. The FREQUENT algorithm uses m counters and determines a set of m candidates that contain all elements that occur more than $n/(m+1)$ times. The output of FREQUENT is therefore a list of elements including all of these heavy users and possibly some light users (*false positives*). To determine all heavy users, Karp *et al.* [12] point out that we cannot do better than keeping a counter for each user, which results in $\Omega(c)$ memory, where c is the number of different users. In the case of Internet packet stream, the number of users (IP addresses) is substantially larger than m. Hence, the algorithm needs much more space than the size of the output. The difficulty of determining all

heavy users is that at any point of time, many users may have nearly equal number of occurrences, and therefore equal chances to be a heavy user, the algorithm must remember the exact count of each user.

In applications like network traffic measurement and accounting, it is important to not only identify all large flows but also to estimate the frequencies of these large flows. Manku *et al.* [13] proposed two algorithms for computing frequencies of all large flows above a user-specified threshold. The *Sticky Sampling* algorithm is probabilistic and with probability $1-\delta$, the algorithm identifies all items whose true frequency exceeds a user specified threshold $s \in (0,1)$ using at most $\frac{2}{\varepsilon}\log(s^{-1}\delta^{-1})$ expected number of counters, where $\varepsilon \in (0,s)$ is the maximum error in the output. The other algorithm called *Lossy Counting* is deterministic in the sense that it outputs all flows above the threshold. Regardless of the threshold s, it achieves the same accuracy ε using at most $\frac{1}{\varepsilon}\log(\varepsilon n)$ counters.

Other work on the particular problem of estimating frequencies in packet streams includes the work of Fang *et al.* [6] who propose heuristics to compute all values above a certain threshold. Charikar *et al.* [2] give algorithms for computing the top k candidates under the Zipf distribution. Estan and Varghese [5] attempt to identify a set of packet classes that are likely to contain the most frequently occurring packet classes, and give probabilistic estimates of the expected count value in terms of a user selected threshold.

1.2 Results of the Paper

In this paper we are concerned with the accuracy of packet counting algorithms. We say that a packet counting algorithm is *k-accurate* if, for any class a that appears n_a times, the algorithm terminates with a counter value c_a for a that satisfies

$$c_a \leq n_a \leq c_a + k \quad . \tag{1}$$

Therefore, both the Sticky Sampling and Lossy Counting algorithms are εn-accurate. We show that the FREQUENT algorithm is $n/(m+1)$-accurate. In general, with m counters, no algorithm is better than $n/(m+1)$ accurate since, if $m+1$ classes each occurs $n/(m+1)$ times then one of those classes will have a counter value of 0 when the algorithm terminates.

However, this argument breaks down when we consider the case when some particular packet class a occurs $n_a \geq \alpha n$ times, for some $\alpha > 1/(m+1)$. In this case, it may be possible for the algorithm to report the number of occurrences of a (and other elements) more accurately. We explore this relationship between accuracy and α. Our results are outlined in the next paragraph.

In Section 2 we show that the FREQUENT algorithm of Demaine *et al.* is $(1-\alpha)n/m$-accurate, where αn is the number of times the most frequently occurring packet class appears in the stream. In Section 3 we give a lower-bound of $(1-\alpha)n/(m+1)$ on the accuracy of any deterministic packet counting algorithm and a lower bound of $(1-\alpha)\Omega(n/m)$ on the accuracy of any randomized packet

counting algorithm. This latter result solves an open problem posed by Demain *et al.* [4] about whether randomized packet counting algorithms can be more accurate than deterministic ones. In Section 4 we summarize and conclude with open problems.

2 The FREQUENT Algorithm

The FREQUENT algorithm of Demaine *et al.* [4] uses m counters. When processing stream item x_i, the following rules are applied in order:

1. If there is a counter monitoring class x_i then increment that counter, otherwise
2. if some counter is equal to 0 then set that counter to 1 and have it monitor class x_i, otherwise
3. decrement all counters by 1.

A nice way to visualize this algorithm is to imagine a set of m buckets that hold colored balls. When a new ball arrives we either place it in the bucket that contains balls of the same color (Case 1), place it in an empty bucket (Case 2) or discard one ball from every bucket as well as the new ball (Case 3).

To analyze the accuracy of this algorithm, we first provide a rough upper-bound on the accuracy and then use this upper-bound to bootstrap a better analysis. Let d be the total number of times Case 3 of the algorithm occurs. No counter is ever less than 0, and each of Case 1 and Case 2 increments exactly one counter. Therefore, if C is the total sum of all counters when the algorithm terminates, then

$$C = n - (m+1)d \geq 0 \ ,$$

so that

$$d \leq \frac{n}{m+1} \ .$$

It follows immediately that for any class a that occurs n_a times, the counter monitoring a has a value of at least

$$c_a \geq n_a - d \geq n_a - \frac{n}{m+1} \ .$$

Suppose that $n_a = \alpha n$ for some $\alpha \geq 1/(m+1)$. Now we can repeat the above argument, since we have just shown that

$$C = n - (m+1)d \geq \alpha n - \frac{n}{m+1} \ ,$$

so that

$$d \leq \frac{(1-\alpha)n}{m+1} + \frac{n}{(m+1)^2} \ .$$

and the value of c_a satisfies

$$c_a \geq \alpha n - d \geq \alpha n - \left(\frac{(1-\alpha)n}{m+1} + \frac{n}{(m+1)^2} \right) \quad .$$

In general, we can repeat the above argument k times to show that

$$c_a \geq \alpha n - \sum_{i=1}^{k} \frac{(1-\alpha)n}{(m+1)^i} - \frac{n}{(m+1)^{k+1}} \quad .$$

In particular, as $k \to \infty$, we obtain $c_a \geq \alpha n - (1-\alpha)n/m$. Now, since c_a is clearly never greater than n_a, we have

$$c_a \leq n_a \leq c_a + \frac{(1-\alpha)n}{m} \quad ,$$

so the output c_a is $(1-\alpha)n/m$-accurate.

Finally, we observe that the above analysis gives an upper-bound on d, and this gives an upper bound on the accuracy of the counter value for a. However, the upper bound on d also gives an upper bound on the accuracy of *any* counter, not just the counter for a. This implies our first result.

Theorem 1 *For any stream in which some element occurs at least αn times, the* FREQUENT *algorithm is $(1-\alpha)n/m$-accurate.*

3 Lower-Bounds on Accuracy

In this section we give lower bounds on the accuracy of deterministic and randomized packet counting algorithms.

3.1 A Deterministic Lower-Bound

Here we give a lower bound for deterministic packet counting algorithms by using an adversary argument. Our adversary builds two distinct streams that the algorithm cannot distinguish between.

Our adversary uses $m+2$ packet classes and builds its streams in two parts (see Figure 1). The first part of both streams is of length $(1-\alpha)n$ and consists of the first $m+1$ packet classes each occurring the same number of times, so that each class occurs $(1-\alpha)n/(m+1)$ times. At this point the two streams diverge. In the first stream, the adversary adds αn occurrences of the unique packet class a of the $m+1$ first classes that is not being monitored by the algorithm after processing the first part of the stream. In the second stream, the adversary adds αn occurrences of the unique packet class z that does not appear in the first part of the stream.

$$\begin{array}{ccccc} abcd\cdots yabcd\cdots y & \cdots & abcd\cdots yaaaaaa\cdots a \\ \underbrace{abcd\cdots y}_{m+1}\underbrace{abcd\cdots y}_{m+1} & \cdots & \underbrace{abcd\cdots y}_{m+1}\underbrace{zzzzzz\cdots z}_{\alpha n} \end{array}$$

Figure 1: The adversary's two streams.

Observe that, since neither a nor z is stored in any of the algorithm's counters after processing the first part of the stream, the only information the algorithm obtains by reading the last element of the stream is that it is not being monitored. Therefore, since the algorithm is deterministic, its counter value c_a for a on the first stream will be equal to its counter value c_z for z on the second stream. However, in the first stream a occurs $n_a = (1-\alpha)n/(m+1)+\alpha n$ times and in the second stream, z occurs $n_z = \alpha n$ times. In order to be accurate at all (refer to (1)) the algorithm must terminate with a counter value $c_a = c_z \leq n_z$. But in this case, the algorithm is not better than $(1-\alpha)n/(m+1)$-accurate for the first stream.

Theorem 2 *For any deterministic algorithm, there exists a stream in which some symbol a occurs $n_a \geq \alpha n$ times, but the algorithm reports a value c_a such that $c_a > n_a$ or $c_a \leq n_a - (1-\alpha)n/(m+1)$.*

3.2 A Randomized Lower-Bound

Next we give a lower bound for randomized algorithms. We do this by providing a probability distribution on input streams such that the expected accuracy of *any* deterministic algorithm on this distribution is at least $(1-\alpha)cn/m$. Since any randomized algorithm is just a probability distribution on deterministic algorithms, the lower-bound therefore holds for randomized algorithms as well.[1] The distribution we use is a probabilistic version of our deterministic construction.

Our distribution uses two constants $1 < c_1 < c_2$ that will be specified later. Each stream of our distribution is a two part data stream made up of $c_2 m$ packet classes. The first part of all streams is identical. As before, it is of length $(1-\alpha)n$, and it consists of the first $c_1 m$ packet classes each occurring an equal number of times, so that each class occurs $(1-\alpha)n/c_1 m$ times. For the second part of the sequence, we select a packet class uniformly at random from all $c_2 m$ classes and make that class occur αn times.

Let a be the packet class chosen to make up the second part of the sequence. Immediately after the first part of the sequence has been processed by the algorithm, there are three cases to consider:

1. The algorithm has a counter that is monitoring a. Since the algorithm has

[1] Technically, this is an application of *Yao's Principle* [14].

only m counters, this happens with probability at most

$$p_1 \leq \frac{m}{c_2 m} = \frac{1}{c_2} \ ,$$

and the number of occurrences of a is $n_1 = (1-\alpha)n/c_1 m + \alpha n$.

2. The algorithm does not have a counter monitoring a and a comes from the first $c_1 m$ packet classes. This happens with probability at least

$$p_2 \geq \frac{(c_1-1)m}{c_2 m} = \frac{c_1-1}{c_2} \ ,$$

and the number of occurrences of a is also $n_2 = (1-\alpha)n/c_1 m + \alpha n$.

3. The class a does not come from the first $c_1 m$ packet classes (so the algorithm is not monitoring a). This happens with probability

$$p_3 = 1 - \frac{c_1}{c_2} \ ,$$

and the number of occurrences of a is $n_3 = \alpha n$.

Let c_a be the value output by the algorithm for class a. Since we are proving a lower-bound, we can assume that in Case 1, the algorithm answers with perfect accuracy, i.e., $c_a = (1-\alpha)n/c_1 m + \alpha n$. However, if the algorithm is not monitoring class a (Cases 2 and 3) then it cannot distinguish between Cases 2 and 3. Since the algorithm is deterministic, it must output the same counter value c_a in both cases. Therefore, the expected error made by the algorithm is at least

$$\begin{aligned}
\mathbf{E}\left[|c_a - n_a|\right] &\geq p_1 \times 0 + p_2 \times |c_a - n_2| + p_3 \times |c_a - n_3| \\
&\geq p_2 \times \left| c_a - \left(\frac{(1-\alpha)n}{c_1 m} + \alpha n \right) \right| + p_3 \times |c_a - \alpha n| \\
&= p_2 \times \left| x_a - \left(\frac{(1-\alpha)n}{c_1 m} \right) \right| + p_3 \times |x_a| \\
&\geq \frac{c_1 - 1}{c_2} \times \left| x_a - \left(\frac{(1-\alpha)n}{c_1 m} \right) \right| + \left(1 - \frac{c_1}{c_2} \right) \times |x_a| \ ,
\end{aligned}$$

where $x_a = c_a - \alpha n$. Setting $c_1 = 1 + \sqrt{2}/2$, $c_2 = 1 + \sqrt{2}$, and simplifying we obtain

$$\begin{aligned}
\mathbf{E}\left[|c_a - n_a|\right] &\geq \frac{\sqrt{2}}{2(1+\sqrt{2})} \times \left(\left| x_a - \left(\frac{(1-\alpha)n}{(1+\sqrt{2}/2)m} \right) \right| + |x_a| \right) \\
&\geq \frac{\sqrt{2}}{2(1+\sqrt{2})} \times \left(\frac{(1-\alpha)n}{(1+\sqrt{2}/2)m} \right) \\
&\geq 0.17157(1-\alpha)n/m
\end{aligned}$$

Theorem 3 *For any randomized algorithm, there exists a stream in which some symbol a occurs $n_a \geq \alpha n$ times, but the algorithm has a counter value c_a such that* $\mathbf{E}\,|n_a - c_a| \geq 0.17157(1-\alpha)n/m$.

We observe that the proof of Theorem 3 extends to a slightly more powerful model in which the packet counting algorithm is allowed to periodically output class/value pairs of the form (a, c_a) whose meaning is "a has occurred c_a times" and the counter value for a is considered to be the last such value output. A similar model is used by Demaine *et al.* [4] to study probabilistic packet streams. To see that the lower-bound carries over, observe that the last such pair (a, c_a) is either output before the second part of the stream begins, or after. In the latter case, the argument above shows that $\mathbf{E}\,[n_a - c_a] = \Omega((1-\alpha)n/m)$. In the former case, the algorithm outputs the value c_a without having seen the final αn occurrences of a. An argument similar to the one above shows that, in this case, there is a packet class a such that $\mathbf{E}\,[n_a - c_a] = \Omega(\alpha n)$.

4 Conclusions

We have studied the problem of approximating the frequency of items in a data stream using a fixed number, m, of counters. We have shown that when some data item a occurs αn times in a stream of length n, then the FREQUENT algorithm of Demaine *et al.* [4] is $(1-\alpha)n/m$-accurate. This is nearly optimal for a deterministic algorithm since we have shown that no deterministic algorithm is better than $(1-\alpha)n/(m+1)$-accurate. Finally, we have shown that randomized algorithms can not be significantly more accurate since any randomized algorithm has an expected accuracy of at least $(1-\alpha)\Omega(n/m)$.

The main open problem left by our research is that of determining if the constant factor in the accuracy of the FREQUENT algorithm can be improved by somehow introducing randomization. It may well be the case that running FREQUENT on a random sample of the original input stream is enough to foil an adversary and improve its accuracy.

References

[1] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the 28th ACM Symposium on the Theory of Computing (STOCS'96)*, pages 20–29, 1996.

[2] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *Proceedings of the 19th International Colloquium on Automata, Languages and Programming*, pages 693–703, 2002.

[3] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2002)*, pages 635–644, 2002.

[4] E. D. Demaine, A. López-Ortiz, and J. I. Munro. Frequency estimation of internet packet streams with limited space. In *Proceedings of the 10th Annual European Symposium on Algorithms (ESA 2002)*, pages 348–360, 2002.

[5] C. Estan and G. Varghese. New directions in traffic measurement and accounting. In *Proceedings of the ACM SIGCOMM Internet Measurement Workshop*, 2001.

[6] M. Fang, S. Shivakumar, H. Garcia-Molina, R. Motwani, and J. Ullman. Computing iceberg queries efficiently. In *Proceedings of the 24th International Conference on Very Large Databases*, pages 299–310, 1998.

[7] M. J. Fischer and S. L. Salzberg. Finding a majority among *n* votes: Solution to problem 81-5 (Journal of Algorithms, june 1981). *Journal of Algorithms*, 3(4):362–380, 1982.

[8] P. Gupta and N. McKeown. Packet classification on multiple fields. In *Proceedings of ACM SIGCOMM*, pages 147–160, 1999.

[9] P. J. Haas, J. F. Naughton, S. Sehadri, and L. Stokes. Samples-based estimation of the number of distinct values of an attribute. In *Proceedings of the 21st International Conference on Very Large Databases (VLDB'95)*, pages 311–322, 1995.

[10] P. Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computations. In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS 2000)*, pages 189–197, 2000.

[11] P. Indyk, S. Guha, M. Muthukrishnan, and M. Strauss. Histogramming data streams with fast per-item processing. In *Proceedings of the 19th International Colloquium on Automata, Languages and Programming*, pages 681–692, 2002.

[12] R. Karp, C. H. Papadimitriou, and S. Shenker. A simple algorithm for finding frequent elements in streams and bags. Unpublished manuscript.

[13] G. Manku and R. Motwani. Approximate frequency counts over data streams. In *Proceedings of the 28th International Conference on Very Large Data Bases*, 2002.

[14] A. C. Yao. Probabilistic computations: Towards a unified measure of complexity. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS'77)*, pages 222–227, 1977.

Prosenjit Bose is with the School of Computing Science, Carleton University, Ottawa, ON, Canada K1S 5B6. E-mail: jit@scs.carleton.ca

Evangelos Kranakis is with the School of Computing Science, Carleton University, Ottawa, ON, Canada K1S 5B6. E-mail: kranakis@scs.carleton.ca

Pat Morin is with the School of Computing Science, Carleton University, Ottawa, ON, Canada K1S 5B6. E-mail: morin@scs.carleton.ca

Yihui Tang is with the School of Computing Science, Carleton University, Ottawa, ON, Canada K1S 5B6. E-mail: y_tang@scs.carleton.ca