Identifying Frequent Items in Sliding Windows over On-Line Packet Streams

Lukasz Golab, David DeHaan, Erik D. Demaine, Alejandro López-Ortiz, and J. Ian Munro

Abstract— **Queries that return a list of frequently occurring items are popular in the analysis of data streams such as real-time Internet traffic logs. In particular, Internet traffic patterns are believed to obey the power law, implying that most of the bandwidth is consumed by a small set of heavy users. While several results exist for computing frequent item queries using limited memory in the infinite stream model, in this paper we consider the limited-memory sliding window model. This model maintains the last N items that have arrived at any given time and forbids the storage of the entire window in memory. We present several deterministic algorithms for identifying frequent items in sliding windows using limited memory and making only one pass over the data, both under arbitrary distributions and assuming that packet types in each instance of the window conform to a multinomial distribution. The former is a straightforward extension of existing techniques and is shown to work well when tested on TCP traffic logs. Our algorithms for the multinomial distribution are shown to outperform classical inference based on random sampling from the sliding window, but lose their accuracy as predictors of item frequencies when the underlying distribution is not multinomial.**

Keywords— **Internet traffic monitoring, on-line stream analysis, sliding windows, frequent item queries.**

I. Introduction

On-line data streams such as TCP/IP packet streams or Web server connection logs possess interesting computational characteristics, such as unknown or virtually unbounded length, possibly very fast arrival rate, inability to backtrack over previously arrived items (only one sequential pass over the data is permitted), and a lack of system control over the order in which data arrive [13]. A particular problem of interest—motivated by traffic engineering, routing system analysis, customer billing, and detection of anomalies such as denial-of-service attacks—concerns statistical analysis of data streams with a focus on newly arrived data and frequently appearing packet types. For instance, an Internet Service Provider may be interested in monitoring streams of IP packets originating from its clients and identifying the users who consume the most bandwidth during a given time interval; see [7], [8] for additional motivating examples. These types of queries, in which the objective is to return a list of the most frequent items (called *top-k queries* or *hot list queries*) or items that occur above a given frequency (called *threshold queries*), are generally known as *frequent item queries*. However, to make such analysis meaningful, bandwidth usage statistics should be kept for only a limited amount of time (for example, one hour or a single billing period) before being replaced with new measurements. Failure to remove stale data leads to statistics aggregated over the entire lifetime of the stream, which are unsuitable for identifying recent usage trends.

Lukasz Golab, David DeHaan, Alejandro López-Ortiz, and J. Ian Munro are with the School of Computer Science, University of Waterloo, Ontario, N2L 3G1, Canada. E-mails: {*lgolab,dedehaan,alopez-o,imunro*}*@uwaterloo.ca*. Erik D. Demaine is with the Massachusetts Institute of Technology (MIT) Laboratory for Computer Science, 200 Technology Square, Cambridge, MA 02139, USA. E-mail: *edemaine@mit.edu*. This research is partially supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada, and by the Nippon Telegraph and Telephone Corporation.

A solution for removing stale data is to periodically reset all statistics. This gives rise to the *landmark window model*, in which a time point (called the landmark) is chosen and statistics are only kept for that part of a stream which falls between the landmark and the current time. Although simple to implement, a major disadvantage of this model is that the size of the window varies—the window begins with size zero and grows until the next occurrence of the landmark, at which point it is reset to size zero. In contrast, the *sliding window model* expires old items as new items arrive. Two common types of sliding windows are *count-based* windows, which maintain the last N packets seen at all times and *time-based* windows, which include only those items which have arrived in the last t time units.

If the entire window fits in main memory, answering threshold queries over sliding windows is simple: we maintain frequency counts of each distinct item within the window and increment or decrement counters as new items arrive and old items are removed. However, in the worst case the entire sliding window may need to be stored in order to identify which packet(s) to expire at any given time. Unfortunately, when monitoring Internet traffic on a backbone link, the packet stream may arrive so fast that useful sliding windows may be too large to fit in main memory (and the system would not be able to keep up with the stream if the window had to be stored on disk). In this case, the window must somehow be summarized and an answer must be approximated on the basis of the available summary information. One solution is to divide each sliding window into sub-windows, store a summary of each sub-window, and re-evaluate the query when the most recent sub-window is full. This reduces space usage, but induces a "jumping window" instead of a gradually sliding window, with the jump size equal to the sub-window size. In practice, the maximum jump size is limited by the answer latency requirements of a particular query or application.

A. Our Contributions

We are interested in identifying frequent items (occurring with a frequency that exceeds a given threshold) in sliding windows over on-line data streams and estimating their true frequencies, while using as little space as possible and making only one pass over the data. We describe three algorithms, MOREFREQUENTITEM, MOSTFREQUENTITEM, and OVERTHRESHOLD, for identifying frequent items within a packet stream adhering to a multinomial distribution in each instance of the window. Our algorithms are deterministic and avoid storing the entire sliding window by operating in the jumping window model, as in the Basic Window approach of Zhu and Shasha [23]. MOREFREQUENTITEM and MOSTFREQUENTITEM—which store only the identity of the item that was most frequent in each Basic Window—identify the most frequent item, but estimating frequency or even bounding the error in the identification is shown to become infeasible as the number of item types grows. In contrast, OVERTHRESHOLD identifies all items over a specified threshold frequency and may be used for frequency prediction with bounded error dependent upon the allocated memory. We demonstrate that MOREFREQUENTITEM and OVERTHRESHOLD outperform classical inference based on random sampling in terms of identifying items over a fixed threshold.

B. Roadmap

The remainder of this paper is organized as follows. Section II presents relevant previous work, Section III describes algorithms MOREFREQUENTITEM and MOSTFREQUENTITEM, while Section IV introduces algorithm OVERTHRESHOLD. Section V compares the prediction error of our algorithms with random sampling and inference for proportions, Section VI concludes the paper with suggestions for future work.

II. PREVIOUS WORK

On-line analysis of network traffic has been one of the primary applications of data stream management systems; examples include Gigascope [5], QuickSAND [12], STREAM [19], and Tribeca [22]. What follows is a brief survey of frequent item algorithms for rapidly arriving packet streams.

A. Frequent Item Algorithms for Infinite Streams

Frequent item algorithms in the infinite stream model employ sampling, counting, and/or hashing to generate approximate answers using limited space. The main difficulty lies in finding a small set of potentially frequent items to monitor, while being able to catch rarely occurring items that suddenly become frequent. In this context, approximation may mean a number of things: an algorithm may either return all of the frequent item types (and some false positives), some frequent item types (and some false negatives), identities of the frequent items but no frequency counts, or identities and approximate counts of the frequent items. Note that the terms *packet types*, *item types*, and *item categories* are used interchangeably throughout the paper.

A naive counting method for answering threshold queries examines all items as they arrive and maintains a counter for each item type. This method takes $\Omega(n)$ space, where n is the number of packets seen so far—consider a stream with $n-1$ unique packet types and one of the types occurring twice. On the other hand, reducing memory usage by random sampling may result in large variance when the sampled frequency is used as the estimator of the actual frequency, especially in the presence of bursty TCP/IP traffic. Estan and Varghese propose a sampled counting algorithm to determine a superset likely to contain the dominant packet types [8]. This algorithm uses random sampling only to select whether an item is to be examined more thoroughly; once an item is selected, all of its occurrences are counted (this idea also appears in Gibbons and Matias [11]). Another counting-sampling approximate algorithm is given by Manku and Motwani in [16], which uses a sampling rate that decreases with time in order to bound memory usage. Finally, a randomized counting-sampling algorithm is presented by Demaine et al. [7] that finds items occurring above a relative frequency of $1/\sqrt{nm}$ with high probability, where n is the number of incoming items observed and m is the number of available counters. This algorithm divides the stream into a collection of rounds, and for each round counts the occurrences of $m/2$ randomly sampled categories. At the end of each round, the $m/2$ winners from the current round are compared with $m/2$ winners stored from previous rounds and if the count for any current winner is larger than the count for a stored category (from any of the previous rounds), the stored list is updated accordingly.

Demaine et al. also present a counting algorithm that uses only m counters and deterministically identifies all categories having a relative frequency above $1/(m+1)$. This algorithm is a straightforward extension of the classical *majority* counting algorithm by Fischer and Salzberg [10]. This method, however, returns a superset guaranteed to contain popular items and requires a re-scan of the data (forbidden in the on-line stream model) to determine the exact set of frequent items. Moreover, Manku and Motwani also show a deterministic counting algorithm that maintains a counter for each distinct item seen, but periodically deletes counters whose average frequencies since counter creation time fall below a fixed threshold. To ensure that frequent items are not missed by repeatedly deleting and re-starting counters, each frequency estimate includes an error term that bounds the number of times that the particular item could have occurred up to now.

Fang et al. present various hash-based frequent item algorithms in [9], but each requires at least two passes over the data. The one-pass sampled counting algorithm by Estan and Varghese may be augmented with hashing as follows. Instead of sampling to decide whether to keep a counter for an item type, we simultaneously hash each item's key to d hash tables and add a new counter only if all d buckets to which a particular element hashes are large (and if the element does not already have a counter). This reduces the number of unnecessary counters that keep track of infrequent packet types. A similar technique is used by Charikar et al. in [3] in conjunction with hash functions that map each key to the set $\{-1, 1\}$. Finally, Cormode and Muthukrishnan give a randomized algorithm for finding frequent items in a continually changing database (via arbitrary insertions and deletions) using hashing and grouping of items into subsets [4].

B. Sliding Window Algorithms

Many infinite stream algorithms, including the frequent item algorithms described above, do not have obvious counterparts in the sliding window model. The fundamental difference is that as new items arrive, old items must be simultaneously evicted from the window, meaning that (at least for some packets) timestamps need to be stored along with packet values.

Zhu and Shasha introduce the concept of *Basic Windows* in order to incrementally compute simple sliding window aggregates [23]. The window is divided into equally-sized Basic Windows and only a synopsis and a timestamp are stored for each Basic Window. When the timestamp of the oldest Basic Window expires, that window is dropped and a fresh Basic Window is added. This scheme works well with statistics that are incrementally computable from a set of synopses. For example, we may incrementally compute the sum of all items inside the current sliding window by replacing the sum of all elements in the oldest Basic Window with the sum of all elements in the newest Basic Window. However, results are refreshed only after the stream fills the current Basic Window. If the available memory is small, then the number of synopses that may be stored is small and hence the refresh interval is large.

In order to solve the above problem, *Exponential Histograms* (EH) are introduced by Datar et al. in [6] for counting the number of ones in a stream consisting of zeros and ones. Given an error bound ϵ, the EH algorithm maintains Basic Windows with exponentially varying size such that the number of windows (and hence, the amount of memory needed for synopses) is optimal. Only a synopsis and a timestamp are stored for each Basic Window. However, in contrast to [23], the EH algorithm returns results to within an error ϵ at all times. The algorithm requires $O(\frac{1}{\epsilon}\log^2 N)$ space, where N is the size of the window, and it is proven that this bound is optimal for counting within the allowed approximation error. EH may only be used with synopses that are mergeable; that is, a synopsis for the union of two Basic Windows must be computable from the two individual synopses. Nevertheless, some statistics that do not obey the additive synopsis property (e.g. variance) may be re-written into an approximate incremental formula (see Babcock et al. [2]). Moreover, Qiao et al. have extended the EH approach to maintain a histogram of values within a sliding window [21].

Finally, random sampling from a window of size N is addressed by Babcock, Datar, and Motwani in [1]. Two algorithms are shown: *Chain Sampling* for count-based windows and *Priority Sampling* for time-based windows.

III. Motivation and Simple Algorithm for Arbitrary Distributions

The two existing techniques for computing statistics over sliding windows (Basic Windows and EH) cannot easily handle top-k queries. For example, if each Basic Window stores counts of the top five categories, we would ignore a frequent category that consistently places sixth. Moreover, the fact that an item type appears in a top-k synopsis in any one Basic Window does not mean that this type is one of the k most frequent types in the entire sliding window (a bursty packet type that dominates one Basic Window may not appear in any other Basic Windows at all). Likewise, the frequent item algorithms for infinite streams do not present obvious opportunities for extension to the sliding window model. The counters used in the counting methods could be split and a timestamp assigned to each sub-counter; this essentially reduces to the Basic Window method with item counts stored in the synopses. Similarly, hash tables could be split in the same way, resulting in a Basic Window approach with hash tables stored in the synopses. Space usage could be improved by incorporating periodic garbage collection to remove infrequent items or items which are about to expire. However, the side effects of the former are that infrequent items that suddenly arrive in large bursts and rise in frequency above the threshold may be missed, while the latter artificially narrows the sliding window.

In [**?**] we proposed the algorithm FREQUENT, that identifies items with frequency higher than a threshold δ with a one sided error, i.e. there are no false positives.

IV. Identifying the Most Frequent Item: Multinomial Distribution

In this section, we present an algorithm for identifying the most frequent item in a sliding window where distinct packet types can be characterized by a multinomial distribution. We show that the algorithm works well when only two categories are present, but becomes computationally expensive as the number of categories increases. In the next section, we modify the algorithm to instead identify items occurring above a given threshold, and we show that solving this problem is significantly easier. We begin by considering count-based windows and extend our approach to time-based windows in Section V-B.1. We continue to assume that reporting the most frequent item need not be available at all times, but instead a slight refresh delay is permitted (as in the jumping window model).

A. *Two Flows*

In the simplest case of only two packet types, call them x and y, whose actual relative frequencies p_x and p_y sum to one, we wish to determine which of the two types occurs in the window with higher frequency and give an estimate for that frequency. Using the Basic Window approach, a simple exact algorithm is to store counters that contain the difference of the number of x-items versus the number of y-items in each Basic Window. Summing the counters over all Basic Windows gives the difference in the observed counts, from which percentage frequencies may be obtained if the size of the sliding window is known. In what follows, we show that if the sliding window conforms to a binomial distribution, we need only record the identity of the more frequent item in each Basic Window in order to estimate item frequencies.

A.1 A Simple Algorithm

The following algorithm divides the sliding window of size N into a set of n equally-sized Basic Windows, each of which is summarized by an entry in a queue. Statistics are refreshed every $b = N/n$ items.

Algorithm MOREFREQUENTITEM

1. Initialize global counters f_x and f_y to zero.
2. Repeat:
   (a) Initialize local counters l_x and l_y to zero.
   (b) For each element e in the next b elements:
      If e is of type x:
         Increment l_x.
      Otherwise:
         Increment l_y.
   (c) Add a summary containing the type of the "winner" (larger local counter) to the back of queue Q, and increment the corresponding global counter.
   (d) If $sizeOf(Q) > N/b$:
      (i) Remove the summary from the front of Q and decrement the corresponding global counter.
      (ii) Output the identity and value of the larger global counter.

Since a single bit can identify the "winner" between two flows, MOREFREQUENTITEM requires $O(N/b)$ space and $\Theta(1)$ amortized time.

Each time a Basic Window is filled, MOREFREQUENTITEM outputs the identity of the item expected to be more frequent in the sliding window. Suppose that the output item is x. The algorithm also supplies a frequency f_x of the Basic Windows dominated by x. However, it is not immediately clear how f_x is related to the actual relative frequency p_x of item x.

Proposition 1: Consider the random variable w defined as follows.

$$w = \begin{cases} 1 & \text{if } x \text{ is the more frequent item in a Basic Window} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Then, w constitutes a Bernoulli variable.

Proof: The probability of success is the same for all Basic Windows as they all have the same size. Success occurs with probability B_x equal to the probability that type x is more frequent than type y within the Basic Window. That is, B_x is the probability that type x occurs $\lceil \frac{b}{2} \rceil$ Or more times in a Basic Window of size b, given by Equation (2); recall that item types x and y conform to a binomial distribution, where the probability that a given element e belongs to type x is p_x and the probability that e belongs to type y is $p_y = 1 - p_x$. Failure occurs with probability $1 - B_x$.

$$B_x = \sum_{i=\lceil \frac{b}{2} \rceil}^{b} \binom{b}{i} p_x^i (1-p_x)^{b-i} \tag{2}$$

■

Corollary 1: Since the probability of type x winning in any one Basic Window is independent of its probability of winning in any other Basic Window, the sum of n Bernoulli variables as defined in Equation (1) is a Binomial variable with parameters n and B_x.

The frequency f_x output by MOREFREQUENTITEM may be used to calculate an observed relative frequency $\hat{B_x}$ that x is the winner of a Basic Window.

$$\hat{B_x} = f_x/n \tag{3}$$

This value can then be substituted in Equation (2) in order to obtain $\hat{p_x}$, the expected relative frequency of item x. Unfortunately, Equation (2) cannot be solved in closed-form for $\hat{p_x}$ (see Appendix for partial results). Thus, numerical methods must be used in order to obtain a value for $\hat{p_x}$ for a given $\hat{B_x}$.

A.2 Bounding the Error

We will make use of the following result due to Hoeffding [14]. Consider a sample of n items from a Binomial distribution and an observed frequency of f. The following is Hoeffding's bound on the deviation of the observed frequency from the true frequency p.

$$\mathbf{Pr}\left\{\frac{f}{n} - p \geq \Delta\right\} \leq e^{-2n\Delta^2} \tag{4}$$

We assume that the numerical methods used to obtain $\hat{p_x}$ from $\hat{B_x}$ are not a significant source of error; therefore, the primary source of error stems from the quality of $\hat{B_x}$ as an estimate for B_x. Now, B_x is a Binomial random variable (by Corollary 1, f_x is a Binomial random variable, and B_x is simply a normalized form of f_x). Using the Hoeffding bound along with a symmetry argument gives the following.

$$\mathbf{Pr}\left\{(\hat{B_x} - \Delta) \leq B_x \leq (\hat{B_x} + \Delta)\right\} > 1 - 2e^{-2n\Delta^2} \tag{5}$$

The right-hand side is the confidence level, so by setting it to the desired confidence (e.g. 0.95) we can solve for Δ (note that n is fixed by the choice of b). Because B_x in Equation (2) increases monotonically with p_x, we can find lower and upper bounds for p_x by numerically computing solutions to Equation (2) for the points $B_x = (\hat{B_x} - \Delta)$ and $B_x = (\hat{B_x} + \Delta)$, respectively. This process is illustrated in Figure 1, showing B_x on the vertical axis and the bounds for p_x on the horizontal axis.

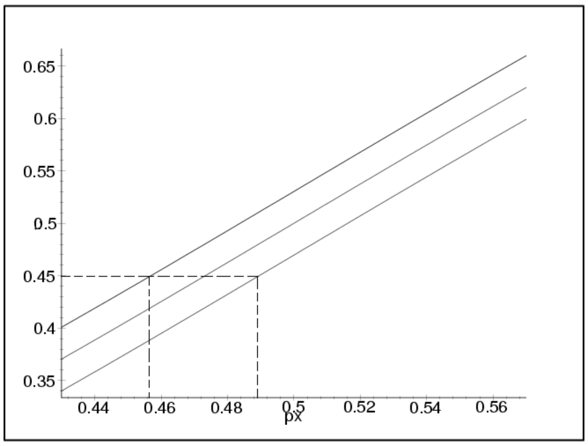


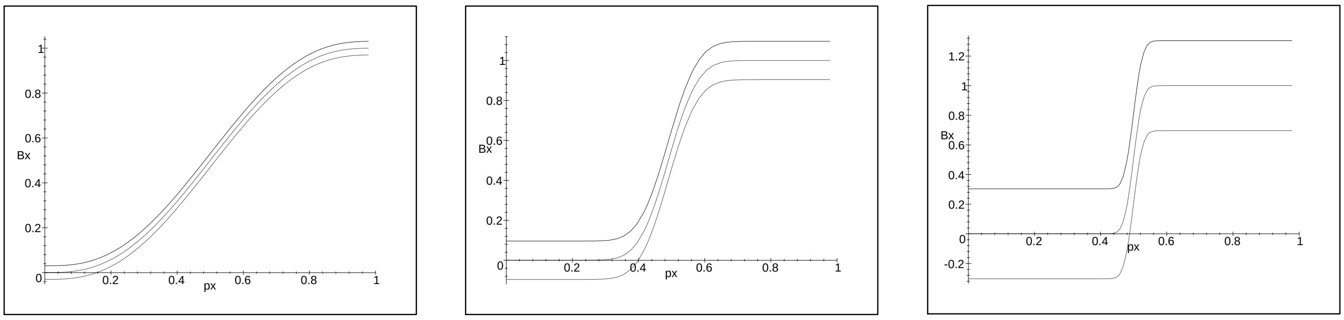
Fig. 1. Solving for $\hat{p_x}$ numerically. The three lines correspond to $\hat{B_x} - \Delta$, $\hat{B_x}$, and $\hat{B_x} + \Delta$. Suppose that the algorithm returns a $\hat{B_x}$ value of 0.45. Then, with 95% certainty the true value of p_x lies in the interval $[0.456, 0.488]$, as shown in the figure.

It should be noted that because the Basic Window size b occurs in the bounds of the summation in Equation (2), the choice of b has a large impact on the error in predicting p_x. As b increases, the following behaviour may be observed.

1. The prediction error Δ surrounding $\hat{B_x}$ increases because n, the number of Basic Windows used to make the prediction, decreases.
2. The graph of $\hat{B_x}$ vs. p_x degrades from a linear function to a step function centered around $p_x = 0.5$.

Figure 2 demonstrates the effect of changing b for a window of size $N = 10000$. It shows the curve $\hat{B_x}$ as a function of p_x, along with the curves $\hat{B_x} - \Delta$ and $\hat{B_x} + \Delta$ that bound the 95% confidence region. The three graphs demonstrate the following values of b: (a) 5 (b) 50 (c) 500.

The observation that $\hat{B_x}$ as a function of p_x degrades to a step function with increasing b is crucial for characterizing the effect of Basic Window size on prediction error. For small values of b, algorithm MOREFREQUENTITEM predicts a wide range of values for p_x, while for large values of b, the useful prediction range for p_x is very small. However, the prediction error immediately



(a) (b) (c)

Fig. 2. Effect of Basic Window size on inference error for $N = 10000$ and (a) $b = 5$, (b) $b = 50$, and (c) $b = 500$.

about the point $p_x = 0.5$ remains tight as b grows. The net effect is that as the choice of Basic Window size ranges from 1 to N, MOREFREQUENTITEM's usefulness as a frequency predictor diminishes, but its accuracy as a Boolean test for identifying the majority item remains. Since the algorithm's space usage is inversely proportional to b, we conclude that there is a direct tradeoff between space and the accuracy of the frequency prediction, but the simple identification of the majority item does not illustrate this tradeoff.

The space requirement of algorithm MOSTFREQUENTITEM consists of two parts: the working space needed to create a summary for the current Basic Window, and the storage space needed for the summaries of the Basic Windows. In the worst case, the working space requires $\min(b, d)$ local counters of size $\log b$. For storage, there are N/b summaries each requiring $\log d$ bits. There are also at most N/b global counters of size $\log(N/b)$. This gives a total space bound of $O(\min(b, d) \log b + \frac{N}{b} \log d + \frac{N}{b} \log \frac{N}{b})$. The time complexity of MOSTFREQUENTITEM is $O(b)$ for each pass through the outer loop. Since each pass consumes b arriving elements, this gives $O(1)$ amortized time per element.

The largest weakness of this algorithm lies in the intractability of using the output value f_i in order to estimate the relative frequency p_i of the most frequent item i. In fact, even just bounding the error on the identity of i is intractable for large d. Consider the case of three item types x, y, and z. In the case of two item types, B_x in Equation (2) was constructed by summing the probabilities of all possible cases where x was in majority within a Basic Window. These cases were easily identified as exactly those where x occurred at least $\lceil \frac{b}{2} \rceil$ times. However, in the case of three categories, the test $count(x) \geq \frac{b}{3}$ is a necessary but not sufficient criterion for identifying a majority by x, because x's majority also depends on its count being greater than both y and z. This gives rise to the equation

$$B_x = \sum_{i=\lceil \frac{b}{3} \rceil}^{b} \binom{b}{i} p_x^i \left\{ \sum_{j=0}^{b-i} \binom{b-i}{j} p_y^j p_z^{b-(i+j)} - \sum_{j=i+1}^{b-i} \binom{b-i}{j} \left[p_y^j p_z^{1-(i+j)} + p_y^{1-(i+j)} p_z^j \right] \right\} \quad (6)$$

with analogous equations existing for B_y and B_z. In order to compute p_x given estimates for B_x, B_y and B_z, we must solve a non-linear system of two equations and two unknowns (the third equation is eliminated by rewriting p_z in terms of p_x and p_y).

In the general case of d packet types, to estimate p_i we must solve a non-linear system of $d-1$ equations and $d-1$ unknowns, where the number of terms within each equation grows combinatorially in d. Even if we restrict the problem to simply bounding the prediction error in the identification of i as the most frequent item, we cannot translate the width of the Hoeffding-bounded error surrounding $\hat{B_i}$ to a range surrounding $\hat{p_i}$ without solving the entire system.

Because the *Most Frequent Item Problem* is a simplification of the more general *Top-k Problem*, the above results demonstrate that it is infeasible to extrapolate a solution to the top-k problem with bounded error using only a set of sub-solutions (top-k lists for portions of the total window) and the assumption of a multinomial distribution of packet types.

V. THRESHOLD QUERIES: MULTINOMIAL DISTRIBUTION

A. The Algorithm

The complexity involved in using algorithm MOSTFREQUENTITEM is due to the interdependence among flows inherent in the concept of a winner for each Basic Window. Because of the dependencies involved in the creation of the stored synopses, we cannot use the synopses to solve for the relative frequency of one flow without simultaneously solving the entire system. Clearly, if we wish to solve for the frequencies of only selected flows, we must eliminate the inter-flow dependencies that exist within the stored synopses.

One way to introduce independence is to replace the concept of *winner* (implying comparison among peers) with *achiever* (implying comparison against an external standard). As a consequence, rather than each Basic Window resulting in exactly one winner, each Basic Window may result in the recognition of zero or more achievers. The following algorithm employs a user-defined threshold $1/m$ to create a synopsis for each Basic Window.

The space complexity of algorithm OVERTHRESHOLD is at most m times worse than that of MOSTFREQUENTITEM, with a worst case bound of $O(\min(b, d) \log b + \frac{mN}{b} \log d + \frac{mN}{b} \log \frac{N}{b})$ where d is the total number of item types in the system. The time complexity is $O(\min(m, b) + b)$ per iteration of the outer loop, which still yields $O(1)$ amortized time.

We now proceed to resolve two issues related to algorithm OVERTHRESHOLD. Firstly, we identify the relation between the frequency f_x output by the algorithm and the true relative frequency p_x. Secondly, we investigate how to calculate the required value of τ used in step 5(d) of the algorithm. We first note that, as in section IV-A, we can define a Bernoulli random variable

$$w = \begin{cases} 1 & \text{if } count(x) \geq b/m \text{ in a Basic Window} \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

whose probability of success B_x is given by the sum of the probabilities for all scenarios where x exceeds the threshold. In the construction of Equation (2) in section IV-A we exploited the fact that "majority" in the case of two categories is equivalent to surpassing a threshold of $\lceil b/2 \rceil$. In this more general case of an arbitrary threshold, our new equation for B_x becomes the following.

$$B_x = \sum_{i=\lceil \frac{b}{m} \rceil}^{b} \binom{b}{i} p_x^i (1 - p_x)^{b-i} \tag{8}$$

Observe that unlike Equation (6), this equation relates B_x to p_x without dependence on the relative frequencies of any other items.

To address the first issue, note that each output f_x induces a value $\hat{B_x} = f_x/n$ which is an approximation for the true B_x of Equation (8). The frequencies $\{f_1, \ldots, f_d\}$ are expected to follow a Multinomial distribution with parameters n, B_1, B_2, $\ldots$, B_d, so the marginal distribution for f_x (and hence $\hat{B_x}$) follows a Binomial distribution. Therefore, we can directly apply the Hoeffding bound from Equation (5) to quantify the error in this approximation. The result is that the observations made in section IV-A.2 regarding the effect of b on the shape of the curve and the error in prediction all directly apply, with the generalization that the step function centers around the point $p_x = 1/m$ rather than $p_x = 1/2$. Figure 3 demonstrates the curve associated with the values $N = 10000$, $b = 50$, and $m = 10$.

The fact that the accuracy of frequency prediction centers around the relative threshold $1/m$ used to create the synopses implies that $1/m$ should be chosen very close to the actual desired reporting threshold. Let us assume that $1/m$ is the desired reporting threshold. Then, the value for τ should be the expected value for B_x when p_x equals the reporting threshold $1/m$, which can be calculated by substituting $1/m$ for p_x in Equation (8). This value for τ gives the most likely list of flows that have a relative frequency over $1/m$; however, the solution may contain either false negatives (high frequency flows not identified) or false positives (low frequency flows incorrectly identified). By adding (subtracting) the value Δ to (from) τ, we can guarantee with the confidence level associated with Δ that the solution contains no false positives (negatives), with the tradeoff that the solution is more likely to contain false negatives (positives).

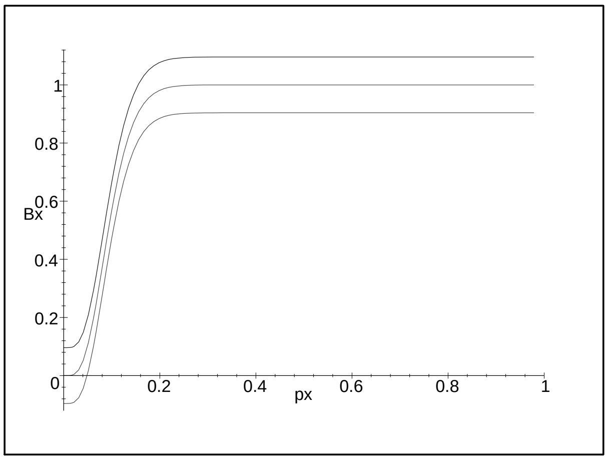


Fig. 3. Effect of threshold parameter on inference error. The three curves correspond to $\hat{B_x} - \Delta$, $\hat{B_x}$, and $\hat{B_x} + \Delta$ for the 95% confidence level.

B. Possible Extensions

B.1 Handling Time-based Sliding Windows

Algorithm FREQUENT does not require uniformly sized Basic Windows, therefore it may be used with time-based windows without any modifications. The other algorithms also work with time-based windows (where Basic Windows of possibly different sizes span equal time intervals) due to the following result from probability theory.

Theorem 1: A Poisson trial a_i is a success with probability p_i and failure with probability $1 - p_i$. Suppose that A is the sum of n independent Poisson trials a_i with probabilities p_i for $1 \leq i \leq n$. Hoeffding's theorem states that A may be upper-bounded by a Binomial random variable B with parameters n and $p = \frac{1}{n} \sum_{i=1}^{n} p_i$.

Unfortunately, Hoeffding's bound for the sum of Poisson trials is known to be (potentially much) looser than the bound on the sum of Bernoulli trials. Alternatively, we may use Chernoff's bound for Poisson trials (see, e.g. [18]).

B.2 Top-k Estimation using Counts

Recall that algorithm OVERTHRESHOLD computes lists of items that occur with frequencies exceeding a user-defined threshold. The following is a possible extension that computes a list of the k most frequent items. Consider the general case of d distinct flows and some threshold τ. In addition to storing the boolean information of whether or not an item exceeded the threshold in a given Basic Window, we also store the counts of all the items above the threshold. After computing the list of all the items that exceed the threshold in the entire window, if there are more than k such items, then we increase the threshold slightly and eliminate all the items whose counts do not exceed the new threshold. We continue this procedure until there are exactly k items left.

The above suggests a more general approach of assigning different thresholds for various item types. That is, for item types x, y, z and w, we could choose to include item x on our above-the-threshold lists only if its relative frequency is above 0.4 and include other items if their frequencies are above 0.35. This would be an appropriate strategy if we knew that x is slightly more popular than the other item types. This method could be improved by incorporating feedback from recent sliding windows and deciding whether to increase or decrease thresholds for various items. This idea, however, is beyond the scope of this paper as it is more akin to probabilistic counting from data synopses than to frequent item queries.

B.3 Reducing Space Usage

We propose two extensions of algorithm OVERTHRESHOLD that reduce space usage: randomly sampling items to be stored in the synopses and deleting parts of older synopses if a particular item has already exceeded the global threshold. In the first approach, if an item exceeds the threshold in a given Basic Window, we flip a biased coin and store the item with probability h and ignore it with probability $1 - h$. This scheme reduces space and does not affect the running time, but it introduces an additional source of error. This demonstrates an interesting tradeoff between using space in order to straighten out the error curve (as in Figure 2, improving the range of p_x that can be predicted) and using space to tighten the prediction error within the usable part of the curve.

The second improvement essentially eliminates redundant information and works as follows. Suppose that an item would have to occur on at least 20 out of 100 top-k lists in order to exceed a given threshold. Suppose further that flow x occurs on 60 such lists. If we removed every second occurrence of flow x from the top-k lists, we would still have 30 such occurrences and we would still conclude that x exceeds the threshold (although we could not even attempt an estimation of the true frequency of x). However, this would introduce error for skewed data as the window slides. A better solution would be to remove flow x from the 30 oldest lists on which it occurs, which does not introduce any error into future windows. In either case, this reduction in space comes at a cost of increased processing time to locate the oldest items to delete.

VI. COMPARISON WITH RANDOM SAMPLING

We are interested in comparing the accuracy in identifying high frequency categories between our algorithms (MOREFREQUENT-ITEM and OVERTHRESHOLD) and classical inference for proportions. Let $\hat{p}$ be the sample proportion (observed count divided by the sample size n). The interval within which the true proportion p lies may be calculated as follows.

$$p \in [\hat{p} - z^*S, \hat{p} + z^*S] \tag{9}$$

The value of z^* is the percentile of the standard normal distribution that corresponds to a given confidence level ($z^* = 1.960$ for 95% confidence, $z^* = 2.576$ for 99% confidence), while S is the standard error of the sample given by the following equation.

$$S = \sqrt{\frac{\hat{p}(1-\hat{p})}{n}} \tag{10}$$

This inference method relies on the normal approximation to Binomial distributions and may be used if $np \geq 5$ and $n(1-p) \geq 5$. Moreover, we introduce the finite population correction factor, which is used when sampling is performed from a finite population. In this scenario, the population size is equal to the sliding window size because we have assumed that each sliding window conforms to a multinomial distribution. With the correction for finite population, assuming sample size n and sliding window (population) size N, the standard error becomes the following.

$$S = \sqrt{\frac{\hat{p}(1-\hat{p})}{n}}\sqrt{\frac{N-n}{N-1}} \tag{11}$$

In our experiments, the error metric is taken to be the maximum expected error when the sample proportion is equal to the threshold at the 95% confidence level. For instance, in the two-flow majority case, we compare the range of p that algorithm MOREFREQUENTITEM returns when $\hat{B_x} = 0.5$ with the confidence interval predicted by Equations (9) and (11) for $\hat{p} = 0.5$. We fix N, the size of the sliding window, to be 10000, and investigate the consequences of increasing the Basic Window size b, (or equivalently, decreasing k, the number of Basic Windows). To ensure fairness, we allow the random sampling algorithm to utilize the same amount of memory that our algorithms require in the worst case. Furthermore, we undercharge the random sampling algorithm by ignoring the space costs associated with maintaining a windowed random sample (see Babcock, Datar, and Motwani [1] for more details regarding these costs).

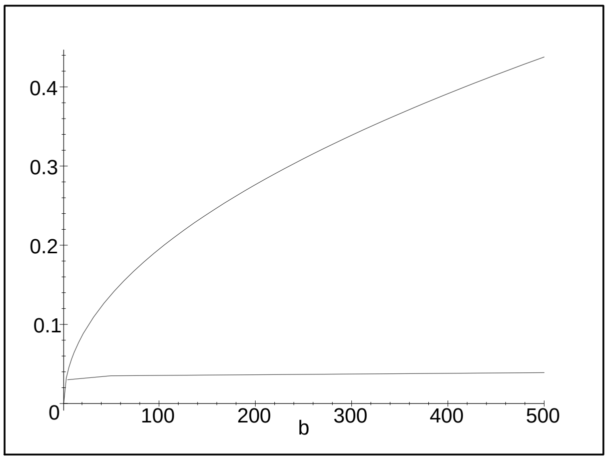


Fig. 4. Prediction error of algorithm MOREFREQUENTITEM (lower curve) and random sampling (upper curve).

A. *Performance of Algorithm* MOREFREQUENTITEM

We begin the performance comparison by considering algorithm MOREFREQUENTITEM in the role of an identifier of the majority between two categories. Figure 4 shows the error (i.e. the length of the interval within which the true value of p_x lies with 95% confidence) as a function of b. The upper curve corresponds to classical inference, the lower curve to MOREFREQUENTITEM.

It can be seen that algorithm MOREFREQUENTITEM outperforms classical inference for all values of b. For instance, if $b = 100$, the algorithm's error is only one-fifth of the error in random sampling. As the value of b approaches 400, our algorithm's advantage in minimizing the error reaches one order of magnitude. As seen in Figure 2 in Section 4, increasing b has little effect on the approximation error of MOREFREQUENTITEM at the decision point, while at the same time reducing the space requirements (and unfortunately, increasing the refresh delay). In contrast, random sampling performs increasingly poorly as b gets large because the ratio of the sample size to the population size decreases. We conclude that MOREFREQUENTITEM is the superior algorithm for the examined parameters.

B. *Performance of Algorithm* OVERTHRESHOLD

We now compare the worst-case performance of algorithm OVERTHRESHOLD with classical inference for many categories with three threshold values: 0.5, 0.1, and 0.01. The value of N remains fixed at 10000 and the confidence level is still 95%. We assume that the number of distinct items d is at least as large as b. Results are shown in Figure 5 for threshold values of (a) 0.5 (b) 0.1 and (c) 0.01. The (approximately) linear function is the error of OVERTHRESHOLD, while the curve corresponds to the error in random sampling.

We first note that the error in classical inference is no longer a monotonically increasing function of b. This is so because the space complexity of algorithm OVERTHRESHOLD depends on b (in the worst case, we need to store the entire current Basic Window in memory since we assumed that $d \geq b$) and on $\frac{mN}{b}$ (the number of synopses stored times the maximum number of items that may possibly exceed the given threshold of $\frac{1}{m}$). Thus, as b increases, our algorithm must allocate more working storage for the current Basic Window, which allows the classical inference algorithm to use a larger sample size. This explains why the error in random sampling eventually begins to decrease as b increases, as seen in Figures 5(a) and 5(b).

Our second observation deals with the degradation in the worst-case performance of algorithm OVERTHRESHOLD (relative to random sampling) for very small threshold values. Clearly, a smaller threshold value allows more items to exceed the threshold in a given Basic Window, thereby increasing the upper bound on the sizes of our synopses. Nevertheless, as seen in Figure 5(a), our algorithm outperforms random sampling when the threshold is 0.5. In Figure 5(b), we see that when the threshold is lowered to 0.1, our algorithm performs better for $b > 25$. In Figure 5(c), further decreasing the threshold to 0.01 leads to a value of $b > 250$ for which algorithm OVERTHRESHOLD is more precise than random sampling.

It should be noted that these results represent the worst-case behaviour of algorithm OVERTHRESHOLD, where the maximal number of items exceeds the threshold in a given Basic Window, and must be recorded in the synopses. Relaxing this condition leads to a relative improvement in the performance of our algorithm versus random sampling. In the "best" case of only two flows, we only require one counter in order to decide which flow was more frequent within the current Basic Window. Thus, the amount of memory available to store a random sample is smaller and our algorithm enjoys a greater relative advantage (see Figure 4).

C. *Performance of Algorithm* MOREFREQUENTITEM

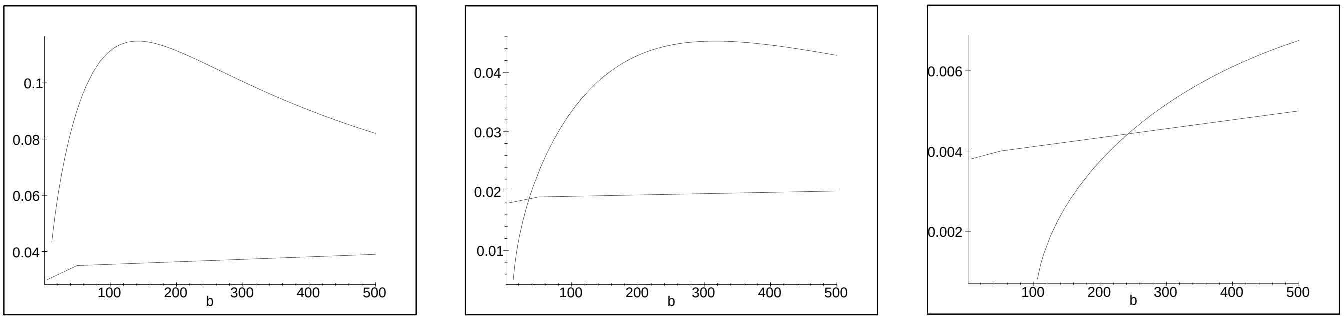
In order to test algorithm MOREFREQUENTITEM, we have edited the TCP log and retained only smtp and ftp packets, which are two of the most frequent protocol types in the trace. Thus, we use the two protocol types as category identifiers rather than source IP addresses as in the previous experiment. The size of the modified trace is 385534 connections. We executed the algorithm over one hundred sliding windows of size 10000 with randomly chosen starting points. Results are shown in Figure 6. The two curves represent the lower and upper frequency prediction ranges for 99% confidence, while the points represent the values of B_x and the corresponding true values of p_x for the two packet types. Figure 6 (a) corresponds to Basic Window size $b = 5$, Figure 6 (b) to $b = 25$, and Figure 6 (c) to $b = 125$. In all cases, the S-shaped prediction curve of our algorithm (which assumes an underlying binomial distribution) does not match the approximately linear prediction curve of the experimental data. In particular, the winning protocol's frequency is consistently underestimated (the observed values lie to the right of the prediction curves) because of the burstiness of the data. That is, if a protocol type wins in a particular Basic Window, it might occur in this window exclusively. On the other hand, the frequency of the losing protocol is often overestimated because the losing protocol may not occur at all in the Basic Windows in which it does not win. We conclude that algorithm MOREFREQUENTITEM should be used (especially as a frequency predictor) only if it is known that the underlying distribution is (or may be approximated as) binomial.

D. *Performance of Algorithm* OVERTHRESHOLD

Results of experiments with algorithm OVERTHRESHOLD are shown in Figure 7. We are again using the full TCP trace and now consider each of the 53 protocol types present in the trace to represent a distinct item type. The sliding window size is 10000. The two curves represent the lower and upper frequency prediction ranges for 99% confidence, while the data points represent the actual (p_x, B_x) pairs generated from the test data. We fix the Basic Window size at 25 and test threshold values of 0.08 (Figure 7 (a)), 0.12 (Figure 7 (b)), and 0.24 (Figure 7 (c)). As before, in many cases the frequencies of the popular packet types are underestimated due to the burstiness of the data. Notably, some of the less frequent protocol types are not as bursty as the most popular types and are well approximated by our multinomial algorithm. Again, we conclude that algorithms MOREFREQUENTITEM and OVERTHRESHOLD should not be used if the underlying data cannot be approximated by a multinomial distribution.

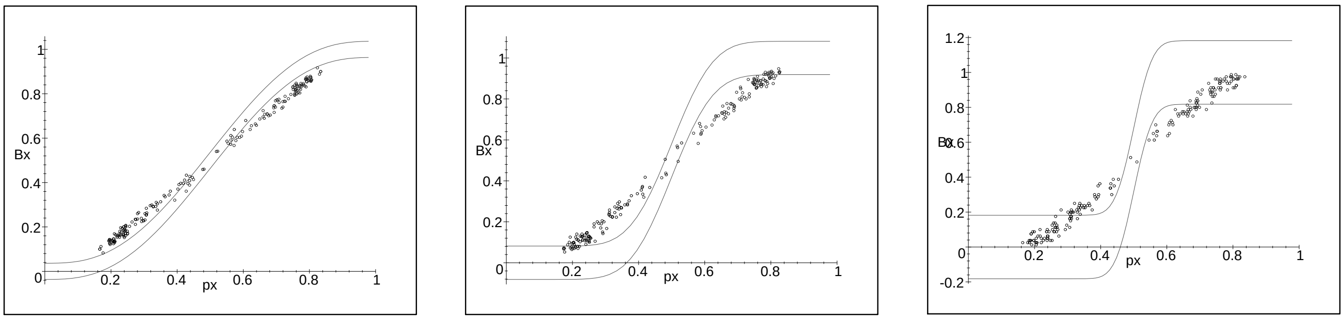
VII. CONCLUSIONS

We presented algorithms for detecting frequent items in sliding windows defined over packet streams. Our algorithms use limited memory (less than the size of the window) and work in the jumping window model. We considered the general case, in which item types conform to an arbitrary distribution and presented a simple algorithm that works well with bursty TCP/IP streams containing a small set of popular item types. We also narrowed down our focus to data conforming to a multinomial distribution and devised algorithms for answering threshold queries, and to some extent for inferring the actual frequencies of items, in this model. These algorithms were shown to outperform classical inference from a windowed random sample,



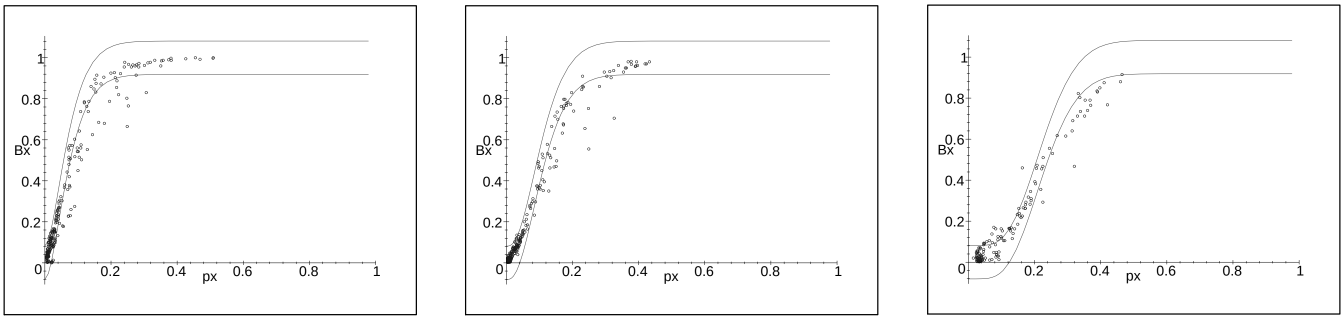
(a) (b) (c)

Fig. 5. Prediction error of algorithm OVERTHRESHOLD (linear function) and random sampling (curve) for three threshold values: (a) 0.5, (b) 0.1, and (c) 0.01.



(a) (b) (c)

Fig. 6. Accuracy of algorithm MOREFREQUENTITEM with our experimental data. The curves represent estimated frequency bounds, while the points show the actual frequencies of item types in the TCP trace. The Basic Window size is 5 in part (a), 25 in part (b), and 125 in part (c).



(a) (b) (c)

Fig. 7. Accuracy of algorithm OVERTHRESHOLD with our experimental data. The curves represent estimated frequency bounds, while the points show the actual frequencies of item types in the TCP trace. The threshold value is 0.08 in part (a), 0.12 in part (b), and 0.24 in part (c).

but performed poorly on the TCP connection stream, in which packet types do not conform to a multinomial distribution.

Our future work includes theoretical analysis of algorithm FREQUENT in order to provide bounds on the probability of false negatives and the relative error in frequency estimation, given a fixed amount of memory and the allowed answer reporting latency. For instance, if the underlying data conform to a power law distribution, we suspect a correlation between k (the size of the synopses required to guarantee some error bound) and the power law coefficient. Another possible improvement concerns translating our results to the gradually sliding window model, where query results are refreshed upon arrival of each new packet. This may be done either by bounding the error in our algorithms due to under-counting the newest Basic Window and over-counting the oldest Basic Window that has partially expired, or perhaps by exploiting the Exponential Histogram approach and its recent extensions in order to extract frequently occurring values. Finally, this work may also be considered as a first step towards solving the more general problem of reconstructing a probability distribution of a random variable given only an indication of its extreme-case behaviour.

REFERENCES

[1] B. Babcock, M. Datar, R. Motwani. Sampling from a Moving Window over Streaming Data. In *Proceedings of the 13th SIAM-ACM Symposium on Discrete Algorithms (SODA)*, 2002, pp. 633–634.

[2] B. Babcock, M. Datar, R. Motwani, L. O'Callaghan. Sliding Window Computations over Data Streams. To appear in *ACM Symp. on Principles of Database Systems (PODS)*, June 2003.

[3] M. Charikar, K. Chen, M. Farach-Colton. Finding frequent items in data streams. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming (ICALP)*, 2002, pp. 693–703.

[4] G. Cormode, S. Muthukrishnan. What's Hot and What's Not: Tracking Most Frequent Items Dynamically. To appear in *ACM Symp. on Principles of Database Systems (PODS)*, June 2003.

[5] C. Cranor, Y. Gao, T. Johnson, V. Shkapenyuk, O. Spatscheck. GigaScope: High Performance Network Monitoring with an SQL Interface. In *Proceedings of the ACM Int. Conf. on Management of Data (SIGMOD)*, 2002, p. 623.

[6] M. Datar, A. Gionis, P. Indyk, R. Motwani. Maintaining Stream Statistics over Sliding Windows. In *Proceedings of the 13th SIAM-ACM Symposium on Discrete Algorithms (SODA)*, 2002, pp. 635–644.

[7] E. Demaine, A. Lopez-Ortiz, J. Ian Munro. Frequency Estimation of Internet Packet Streams with Limited Space. In *Proceedings of the 10th European Symposium on Algorithms (ESA)*, 2002, pp. 348–360.

[8] C. Estan, G. Varghese. New Directions in Traffic Measurement and Accounting. In *Proceedings of the ACM SIGCOMM Internet Measurement Workshop*, 2001, pp. 75–80.

[9] M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, J. Ullman. Computing Iceberg Queries Efficiently. In *Proceedings of the 24th Int. Conf. on Very Large Databases (VLDB)*, 1998, pp. 299–310.

[10] M. Fischer and S. Salzberg. Finding a majority among N votes: Solution to problem 81-5 (Journal of Algorithms, June 1981). In *Journal of Algorithms*, 3(4):362–380, December 1982.

[11] P. Gibbons, Y. Matias. New Sampling-Based Summary Statistics for Improving Approximate Query Answers. In *Proceedings of the ACM Int. Conf. on Management of Data (SIGMOD)*, 1998, pp. 331–342.

[12] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, M. J. Strauss. QuickSAND: Quick Summary and Analysis of Network Data. DIMACS Technical Report 2001-43, Dec. 2001. Available at `http://citeseer.nj.nec.com/gilbert01quicksand.html`.

[13] L. Golab, M. T. Ozsu. Issues in Data Stream Management. To appear in *SIGMOD Record*, Volume 32, Number 2, June 2003. Extended version available at `http://db.uwaterloo.ca/~ddbms/publications/stream/streamsurvey.pdf`.

[14] W. Hoeffding. Probability Inequalities for Sums of Bounded Random Variables. In *American Statistical Association Journal*, 58:13–30, 1963.

[15] The Internet Traffic Archive. `http://ita.ee.lbl.gov`.

[16] G. Manku, R. Motwani. Approximate Frequency Counts over Data Streams. In *Proceedings of the 28th Int. Conf. on Very Large Data Bases (VLDB)*, 2002, pp. 346–357.

[17] Maple Version 8, `http://www.maplesoft.com`.

[18] R. Motwani, P. Raghavan. Randomized Algorithms. Cambridge University Press, 1995.

[19] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, R. Varma. Query Processing, Approximation, and Resource Management in a Data Stream Management System. In *Proceedings of the 1st Biennial Conf. on Innovative Data Syst. Res (CIDR)*, 2003, pp. 245–256.

[20] V. Paxson, S. Floyd. Wide-Area Traffic: The Failure of Poisson Modeling. In *IEEE/ACM Transactions on Networking*, 3(3):226–244, June 1995.

[21] L. Qiao, D. Agrawal, A. El Abbadi. Supporting Sliding Window Queries for Continuous Data Streams. To appear in *15th Int. Conf. on Scientific and Statistical Database Management (SSDBM)*, July 9-11, 2003.

[22] M. Sullivan, A. Heybey. Tribeca: A System for Managing Large Databases of Network Traffic. In *Proceedings of the USENIX Annual Technical Conf.*, 1998.

[23] Y. Zhu, D. Shasha. StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time. In *Proceedings of the 28th Int. Conf. on Very Large Data Bases (VLDB)*, 2002, pp. 358–369.

APPENDIX

RELATING B_x AND p_x

The following calculations have been performed in Maple 8.00 [17]. Here, we show that the closed form solution of Equation (2) is impractical to compute. We begin by restating Equation (2), that is, the probability of flow x winning a particular Basic Window in the case of only two flows.

$$B_x = \sum_{i=\lceil \frac{b}{2} \rceil}^{b} \binom{b}{i} p_x^i (1-p_x)^{b-i} \tag{12}$$

Solving the summation in Equation (12), we obtain

$$\begin{aligned} B_x \;=\; & \binom{b}{\lceil \frac{b}{2} \rceil} p_x^{\lceil \frac{b}{2} \rceil} (1-p_x)^{b-\lceil \frac{b}{2} \rceil} \\ & H\left([\lceil \frac{b}{2} \rceil - b, 1], 1 + \lceil \frac{b}{2} \rceil, \frac{p_x}{p_x - 1} \right) \end{aligned} \tag{13}$$

where the generalized Hypergeometric function $H([n_1, ..., n_j], [d_1, ..., d_m], z)$ is defined as

$$H(\mathbf{n}, \mathbf{d}, z) = \sum_{k=0}^{\infty} \frac{\prod_{i=1}^{j} \frac{\Gamma(n_i+k)}{\Gamma(n_i)} z^k}{\prod_{i=1}^{m} \frac{\Gamma(d_i+k)}{\Gamma(d_i)} k!} \tag{14}$$

where the Gamma function $\Gamma(z)$ is

$$\Gamma(z) = \int_0^{\infty} e^{-t} t^{z-1} dt \tag{15}$$

Maple is unable to analytically solve for p_x in Equation (12). This is also the case for multiple flows and an arbitrary threshold—the only difference is that $\lceil \frac{b}{2} \rceil$ is replaced by τb where $0 \leq \tau \leq 1$ is the threshold.