Frequency Estimation of Internet Packet Streams with Limited Space*

Erik D. Demaine[1], Alejandro López-Ortiz[2], and J. Ian Munro[2]

[1] Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139, USA, `edemaine@mit.edu`

[2] School of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada, {`alopez-o,imunro`}`@uwaterloo.ca`

Abstract. We consider a router on the Internet analyzing the statistical properties of a TCP/IP packet stream. A fundamental difficulty with measuring traffic behavior on the Internet is that there is simply too much data to be recorded for later analysis, on the order of gigabytes a second. As a result, network routers can collect only relatively few statistics about the data. The central problem addressed here is to use the limited memory of routers to determine essential features of the network traffic stream. A particularly difficult and representative subproblem is to determine the top k categories to which the most packets belong, for a desired value of k and for a given notion of categorization such as the destination IP address.

We present an algorithm that deterministically finds (in particular) all categories having a frequency above $1/(m+1)$ using m counters, which we prove is best possible in the worst case. We also present a sampling-based algorithm for the case that packet categories follow an arbitrary distribution, but their order over time is permuted uniformly at random. Under this model, our algorithm identifies flows above a frequency threshold of roughly $1/\sqrt{nm}$ with high probability, where m is the number of counters and n is the number of packets observed. This guarantee is not far off from the ideal of identifying all flows (probability $1/n$), and we prove that it is best possible up to a logarithmic factor. We show that the algorithm ranks the identified flows according to frequency within any desired constant factor of accuracy.

1 Introduction

Problem. The goal of this research is to develop algorithms that extract essential characteristics of network traffic streams passing through routers, specifically estimates of the heaviest users and most popular sites, subject to a limited amount of memory about previously seen packets. Such characteristics are essential for designing accurate models and developing a general understanding of Internet

* This research is partially supported by the Natural Science and Engineering Research Council of Canada, by the Canada Research Chairs Program, and by the Nippon Telegraph and Telephone Corporation through the NTT-MIT research collaboration.

traffic patterns, which are important for such applications as efficient network routing, caching, prefetching, information delivery, and network upgrades. In addition, information of the load distribution has direct applications to billing users.

As the network stream passes by, we have only a few nanoseconds to react to each packet. This time permits, at best, indexing into one of a small number of registers and storing a new value or incrementing or decrementing a few counters. Memory is limited primarily because it must be on the chip that is handling our processing, in order to keep up.

Ideally, we would like to determine the heaviest k users, for a desired value of k, over some time period. However, because some users may have nearly equal load, answering this question exactly is impossible using little space. Rather, one problem we consider is to determine all users above a given load threshold during some time period. A second case of interest is the weaker requirement of identifying a short list of elements guaranteed to include all of these heavy users. Of course, we would like to be able to solve these problems in the worst case for all possible input sequences, but failing that, we may settle for a probabilistic method provided it is robust (accurate with high probability).

Application. In practice, this frequency estimation information is used both for billing purposes and for traffic engineering decisions. In our particular case, this research is motivated by the need to determine the largest packet flows which most heavily influence the characteristics of a router. The routers in question serve large capacity connections on backbones across the continental United States. In network-administration parlance, we need to determine the flows that "shape" the pipe. The information collected in this scenario is important for short- and long-term traffic engineering and routing decisions on the pipe.

In this application, we augment the router by adding a monitoring system to the router box that collects aggregate statistics on the traffic. This system monitors the packet stream as it passes by, and must collect statistical data in real time. Given the current bandwidth capacities at the network core, the processing time must be on the order of nanoseconds for each packet. This imposes particular restrictions in the nature and amount of operations that can be performed per packet, usually limited to manipulating a small number of registers. Often we can assume the existence of a hardware-based hash-table (associative memory). This table implements a hardware lookup operation using only a few clock cycles. It returns an index associated with the entry if present or an error flag otherwise.

As an example, routers from one of the largest vendors (Cisco) collect perfect statistics on low-bandwidth connections but rely on sampling for higher speeds. The following excerpt from the Cisco *NetFlow* manual [5] illustrates this:

> Forwarding rates on a Gigabit Switch Router...an order of magnitude greater than traditional platforms that support NetFlow. "Touching" every switched packet for NetFlow accounting becomes a challenge at these high switching rates. However, collecting characteristic statistics on IP traffic being forwarded...is still a necessary tool for managing and planning a network.

In order to scale to higher forwarding rates, NetFlow will now allow the user to sample one out of every "x" IP packets being forwarded... This feature will substantially decrease the CPU utilization needed to account for NetFlow packets.

However, this sampling method is often unsatisfactory given the nature of Internet traffic [9, 23]. Moreover, in many cases, a small percentage of the packet categories account for a large percentage of the traffic. In general, because of the nature and characteristics of Internet traffic and intended routing application, we require counting mechanisms that examine the vast majority of packets using contiguous sampling of packet bursts.

Our results. We consider a general model in which packets have been classified into *categories*. Examples of interesting categorizations include the IP address and/or port of the packet's source and/or destination. We illustrate under a variety of weak models of computation, storage, and network distributions that carefully arranged counting of repetitions of packets' categories can lead to accurate estimates of the most common packet categories above a certain threshold. To give some intuition, a representative example of how counters can be used is the following: when a packet streams by, the process can check whether its category matches any of the currently monitored categories, and if so, increment that counter. The idea is that the category with the highest counter is likely to be the most popular category.

The primary difficulty in counting with very few counters is to know which categories to monitor. If we never reset the counters and start counting newly discovered categories, we may never notice the most popular category, thus never counting them and discovering their popularity. On the other hand, if we reset counters too frequently, we will not gain enough statistics to be sure which counter is significantly higher than the others.

We resolve this trade-off with the following matching upper and lower bounds for monitoring a stream of *unknown* length using m counters:

1. In the worst-case omniscient-adversary model [Section 3]:
   (a) All categories that occur more than $1/(m+1)$ of the time can (in particular) be deterministically reported after a single pass through the stream. However, it is unknown which reported categories have this frequency.
   (b) This result is best possible: if the most common category has frequency of less than $1/(m+1)$, then the algorithm can be forced to report only uniquely occurring elements.
2. In the stochastic model [Section 4]:
   (a) All categories that occur with relative frequency $> (c \ln n)/\sqrt{mn}$ for a constant $c > 0$ can be reported after a single pass through the stream.
   (b) The algorithm estimates the frequencies of the reported categories to within a desired error factor $\varepsilon > 0$ (influencing c).
   (c) The results hold *with (polynomially) high probability*, meaning that the probability of failure is at most $1/n^i$ for a desired constant i (also influencing c).

(d) This result is best possible up to constant factors: if the maximum frequency is below $f/\sqrt{nm}$, then the algorithm can be forced to report only uniquely occurring elements with probability at least $(e^{-1+1/e})^f$.

3. Both of these one-pass algorithms can be implemented in a small constant amount of worst-case time per packet.

Related work. Some variants of this problem have been previously considered in the context of one pass analysis of database streams [1, 10, 20], query streams to a search engine [3], and packet data streams [7, 9, 19, 21]. Morris [24] showed that it is possible to approximately count up to n using $\lg\lg n$ bits, and Flajolet [15] gave a detailed analysis of this algorithm. Vitter [26] shows how to sample in a small amount of space and linear time in a single pass. A related problem is computing the spectra (approximate number of distinct values) of a stream which can be achieved in $\lg n$ space [16, 27]. Alon et al. show that the first five moments can be approximated in $\lg n$ space while surprisingly all other (higher) moments require linear space [1].

On the particular issue of estimating frequencies, Fang et al. [10] propose heuristics to compute all values above a threshold. Charikar et al. [3] propose algorithms to compute the top k candidates in a list of length l under a Zipfian distribution. Estan and Varghese [9] identify supersets likely to contain the dominant flows and give a probabilistic estimate of the expect count value in terms of a user selected threshold.

2 Model

This section formalizes the problems and models addressed in this paper, some aspects of which were mentioned in Section 1 in the context of our application. There are three key aspects to the problem and model: what computational power and storage we have to gather statistics about streams, what distributions the streams follow, and what guarantees we make about quality of results. We cover each aspect in the next three subsections.

2.1 Computation and Storage

We use a more restrictive model for the algorithms we develop, and a more powerful model for proving lower bounds, strengthening our results.

2.1.1 Model for Algorithms. Our basic model of computation is that a *statistics-gathering* process watches a stream of n packets passing through an Internet router or similar device. The stream is rapid, so the process can make only one pass through the data, and furthermore can perform little computation per packet. Specifically, we limit the amount of computation to $O(1)$ operations per packet. The storage space available to the process is limited, but a more important limiting factor is that the *working store* of the process is very small: all actively used variables (e.g., counters) must fit in a small cache in order to keep up with the data stream. Thus, in some settings, we may be willing to record a significant amount of data (but still much less than one item per packet) to external storage, and make a final pass through these records at the end of the computation.

A key operation that the statistics gathering process can perform is *counting*. The process is limited to having at most m active counters at any time. Each counter has an associated category that it *monitors*. A counter can be incremented, decremented, or reset to monitor a different category.

Counters can be associatively indexed based on the monitored category. This indexing structure can be implemented in hardware by associative memory, or in software using dynamic perfect hashing [25]. In the latter case, our worst-case running times turn into with-high-probability running times.

We believe that this model of computation captures essentially the entire spectrum of possible algorithms, while capturing all of the important limiting factors in the application. For lower bounds, however, we will consider an even more powerful model, described next.

2.1.2 Model for Lower Bounds. For the purpose of lower bounds, we consider a broad model of computation in which the process can maintain at most m categories in working store at any time, in addition to examining the category of the current packet under consideration. Arbitrary amounts of memory and computation can be used for counters or other structures, but categories must be treated as opaque objects from an arbitrary space with unknown structure, and at most m categories can be stored. The only operation allowed on categories is testing two for equality; in the lower-bound context where we ignore computation time, this operation permits hashing based on categories currently in working store. The process can return candidate most-popular categories only from the m categories that it has in working store.

2.2 Network Traffic Distributions

We propose three broad models of the network traffic distributions that enable us to prove guarantees on quality. All of these models lead to interesting theoretical results which are closely related to the practical problem.

The two most general models are *worst-case distributions*. In this context, the network traffic is essentially arbitrary, and at any moment, an adversary can choose the next packet's category. Algorithms in this model are difficult but surprisingly turn out to be possible. There are two subtly different versions of the model. In the *omniscient adversary* model, the adversary knows everything about the algorithm's execution, and can choose the packet sequence to be the absolute most difficult. In the slightly less powerful but highly natural *oblivious adversary* model, the adversary knows the entire algorithm, but does not know the results of any random coin tosses made by the algorithm. Thus the algorithm can hope to win over the adversary with high probability by using random bits.

Of course, these worst-case models are overly pessimistic, and limit the provable strength of any algorithm. Fortunately, real traffic is not worst-case, but rather follows some sort of distribution. A natural such distribution is the *stochastic* model: an arbitrary probability distribution specifies the relative frequencies of the category, but in what order these categories occur in the packet stream is uniformly random. While this model may not precisely match reality, we feel that it is sufficiently representative to lead to highly practical algorithms. (We plan to evaluate this statement experimentally.)

2.3 Guarantees

It is impossible in general to report the most common category in one pass using less than $\Theta(n)$ storage. For example, such storage is clearly necessary when all categories occur uniquely except for one category that occurs twice. Fortunately, a user of this system is only interested in categories that occur particularly often, i.e., above some frequency threshold.

It turns out that, for each model of network traffic, there is a particular threshold below which it is impossible to accurately detect, but above which it is possible to accurately detect. When we have no extra storage beyond the working store, we can only report m such categories with any confidence. When we have extra storage beyond the working store, we can record more values and make a final pass to choose the largest k frequencies for a desired value of k. In either case we guarantee that, out of the categories whose frequencies are above threshold, the approximately top k are reported. "Approximately" means that the frequency (as opposed to rank) is within a desired constant-factor error.

3 Worst-Case Bounds without Randomization

This section develops an algorithm for the most difficult model, the worst-case omniscient adversary.

3.1 Classic Majority Algorithm

Our starting point is the elegant algorithm [13] for determining whether a value occurs a majority of the time in a stream, i.e., occurs more than $n/2$ times in a stream of length n. The basic model under which this algorithm was developed is that we should make as few passes as possible through the data and as few comparisons as possible, while using the smallest possible amount of space—a single counter.

```
Algorithm MAJORITY
 1. Initialize the counter to zero.
 2. For each element in the stream:
    (a) If the counter is zero, define the current element to be the monitored
        element of the counter.
    (b) If the current element is the monitored element, increment the counter.
        Otherwise, decrement the counter.
```

If the algorithm terminates with a counter value of zero, then the last monitored element or the last value on the stream could have occurred up to $n/2$ times, though not a majority. On the other hand, if the counter value is positive, the last monitored element is the only value that could have occurred in a majority of the positions. A simple rescan (not permitted in our model) confirms or denies the hypothesis, although Fischer and Salzberg [13] present the method somewhat differently and reorder the elements in order to achieve the optimal worst case bound of $\lceil 3n/2 \rceil - 2$.

3.2 Generalization

This majority algorithm is a gem, often used in undergraduate lectures and assignments. However, the following generalization does not seem to have appeared. Our initial description ignores issues of data structures required to effectively decrement m counters at once or manage any other aspects of the algorithm; these issues will be addressed later.

Theorem 1. *There is a single-pass algorithm using m counters that determines a set of at most m values including all that occur strictly more than $n/(m+1)$ times in an input stream of length n.*

Proof. The scheme is indeed a generalization of Algorithm Majority:

```
Algorithm Frequent
 1. Initialize the counters to zero.
 2. For each element in the stream:
    (a) If the current element is not monitored by any counter and some counter
        is zero, define the current element to be the monitored element of that
        counter.
    (b) If the current element is the monitored element of a counter, increment
        the counter. Otherwise, decrement every counter.
```

The reaction to a value not in a full slate of candidates is admittedly Draconian, but it is effective. To demonstrate this effectiveness, consider any element x that occurs $t > n/(m+1)$ times. Suppose that x is read t_f times when all other candidate locations are full with other values, and t_i times when either it is already present or there is space to add it. Thus, x's counter is incremented t_i times, and $t_f + t_i = t > n/(m+1)$. Furthermore, let t_d denote the number of times that a counter monitoring x is decremented as another value is read. Because a counter never goes negative, $t_i \geq t_d$. If this inequality is strict, then x ends up with a positive count at the end of the algorithm.

With each of the $t_f + t_d$ times decrements occur, we can associate m occurrences of other values along with the occurrence of x, for a total of $m+1$ unique locations in the input steam. Thus, $(m+1)(t_f + t_d) \leq n$. If the final value of x's counter is zero, then $t_d = t_i$, so $t = t_f + t_i = t_f + t_d > n/(m+1)$, i.e., $(m+1)(t_f + t_d) > n$, which is a contradiction. Hence $t_i > t_d$, so x's counter remains positive and x is one of at most m candidates remaining. □

This method thus identifies at most m candidates for having appeared more than $n/(m+1)$ times, and does so with no use of probabilistic methods. Clearly there remains the issue of how to perform the appropriate updates quickly. Most notably, there is the issue of decrementing and releasing several counters simultaneously.

3.3 Data Structures

To support decrementing all counters at once in constant time, we store the counters in sorted order using a differential encoding. That is, each counter actually only stores how much larger it is compared to the next smallest counter. Now incrementing and decrementing counters requires them to move significantly in the total order; to support these operations, we coalesce equal counters (differentials of zero) into common *groups*.

The overall structure is a doubly linked list of groups, ordered by counter value. Each group represents a collection of equal counters, consisting of two parts: (1) a doubly linked list of counters (in no particular order, because they all have the same value), and (2) the difference in value between these counters and the counters in the previous group, or, for the first group, the value itself. Each "counter" no longer needs to store a value, but rather stores its group and its monitored element.

Because of lack of space, we omit the details of Algorithm FREQUENT in combination with these data structures.

Theorem 2. *Algorithm* FREQUENT *can be augmented to run in in $O(1)$ time per packet.*

3.4 Lower Bound

Algorithm FREQUENT achieves the best possible frequency threshold according to the model presented in Section 2.1.2.

Theorem 3. *For any n and m, and any deterministic one-pass algorithm storing at most m elements at once, there is a sequence of length n, in which one element occurs at least $n/(m+1)-1$ times and the other elements are all unique, and on which the algorithm terminates with only uniquely occurring elements stored.*

Proof. We initially imagine there being n distinct elements, divided by a yet-to-be-determined scheme into $m+1$ *classes*. We maintain that each element stored by the algorithm is from a different class. At each step, the algorithm examines its at most $m+1$ elements, discards one, and reads the next element from the stream. The adversary chooses the next element from the same class as the element that was discarded. (At the beginning, the adversary chooses arbitrarily.)

In this way, the algorithm learns only that elements from different classes are different, but does not learn about elements from a common class. Thus, at the end, the adversary is free to choose which elements in a class are equal and which are not. In particular, the adversary can choose the largest class, which must have size at least $n/(m+1)$, to have all its members equal except for possibly one member of the m being returned by the algorithm; and choose all other classes to have all distinct elements. $\square$

4 Probabilistic Frequency Counts

This section develops algorithms for the stochastic model, in which an arbitrary probability distribution specifies the relative frequencies of the categories, but in what order these categories occur in the packet stream is uniformly random. We distinguish two cases according to whether the process is allowed extra storage so long as the working store is small; see Section 2.1.1.

4.1 Overview

The basic algorithm works as follows. We divide the stream into a collection of rounds, carefully sized to balance the counter-reset trade-off described in the first section. At the beginning of each round, the algorithm samples the first m distinct packet categories, which is equivalent to sampling m packets uniformly at random. The algorithm then counts their occurrences for the duration of the round. Applying Chernoff bounds, we prove that the counts obtained during a round are close to the actual frequencies of the categories. The k categories with the maximum counter values at the end of the round are the winners for that round. If extra *nonworking storage* is available to the algorithm, we record these winners and their counts for a final tournament at the end of the algorithm. Otherwise, we reserve a constant fraction of the working storage for the current best winners, and only compare against those. In either case, we prove that with high probability the true frequencies of the final winners are close to the frequencies of the truly most popular categories. The probabilities are slightly higher when extra nonworking space is available.

The ideal choice for the size of a round in this algorithm depends on the length n of the stream and on the probability distribution on categories. Of course, the algorithm does not generally know the probabilities, and may not even know for how long it will be monitoring the stream: imagine a scenario in which the statistics gathering process is running constantly, and at will a networks designer can request the current guess and confidence of the most popular categories; as time passes, the confidence increases. To solve these problems, we harness the algorithm in an adaptive framework that gradually increases the round length until the confidence is determined to suffice. This flexible framework requires monitoring the stream for only slightly longer.

4.2 Algorithm with Extra Nonworking Storage

More precisely, we divide the input stream into rounds of r packets each. The algorithm works as follows and the theorem follows from a careful examination of Chernoff bounds.

```
Algorithm PROBABILISTIC
 1. For each round of $r$ elements:
    (a) Assign the $m$ counters to monitor the first $m$ distinct elements that
        appear in the round.
    (b) For each element, if the element is being monitored, increment the ap-
        propriate counter.
    (c) Store the elements and their counts to the extra nonworking storage.
 2. Pass through the elements and counts stored in extra nonworking storage.
 3. Return the $k$ distinct elements with the largest counts, for the desired value
    of $k$. (If an element appears multiple times in the list, we effectively drop all
    but its largest count.)
```

Theorem 4. *Fix any constants $c > 0$ and $\alpha > 1$. Call an element* above threshold *if it has relative frequency at least $\tau = (c \ln n)/\sqrt{mn}$. Suppose that t elements are above threshold. If c is sufficiently large with respect to α, then with high probability, Algorithm* PROBABILISTIC *with $r = \sqrt{mn}$ returns a list of k elements whose first* $\min\{k, t\}$ *elements are as if we perturbed each element's relative frequency within a factor of α and then took the top* $\min\{k, t\}$ *elements.*

4.3 Algorithm without Extra Nonworking Storage

A simple modification to Algorithm PROBABILISTIC avoids the use of extra storage by computing the maximum frequencies online at the cost of using some counter space:

```
Algorithm PROBABILISTIC-INPLACE
 1. Reserve $m/2$ of the $m$ counters to store the current best candidates.
 2. For each round of $r$ elements:
    (a) Assign the $m/2$ unreserved counters to monitor the first $m/2$ distinct
        elements that appear in the round, and zero these counters.
    (b) For each element, if the element is being monitored, increment the ap-
        propriate counter.
    (c) Replace the $m/2$ reserved counters with the top out of all $m$ counters.
 3. Return the $m/2$ reserved counters.
```

As stated, this algorithm does not run in constant time per packet, incurring a $\Theta(m)$ cost at the end of every round. However, this large cost can be avoided, similar to Algorithm FREQUENT. Again we omit details because of lack of space.

We obtain the same results as in Theorem 4, only with m half as large and k constrained to be at most $m/2$.

Theorem 5. *Suppose that t elements are above threshold, i.e., have relatively frequency at least $(c \ln n)/\sqrt{mn/2}$. If c is sufficiently large with respect to α, then with high probability, Algorithm* PROBABILISTIC-INPLACE *and its enhancement with $r = \sqrt{mn/2}$ return a list of $m/2$ elements whose first $\min\{m/2, t\}$ elements are as if we perturbed each element's relative frequency within a factor of α and then took the top $\min\{m/2, t\}$ elements.*

4.4 Streams of Unknown Length

If the value of n is unknown to the algorithm, we can guess the value of n to be 1 and run the algorithm, then guess consecutively $n = 2, 4, \ldots, 2^j, \ldots$ until the stream is exhausted. At round j, we can find the top elements so long as their probability satisfies $p > j/\sqrt{2^j m}$. This bound is within a factor of roughly $\sqrt{2}$ compared to if we knew n a priori.

4.5 Lower Bound

We can prove a matching lower bound for the algorithms above, up to constant factors, in the model of computation presented in Section 2.1.2:

Theorem 6. *Consider the distribution in which one element x has relative frequently (just) below $f/\sqrt{mn}$, and e.g. every other element occurs just once. For any probabilistic one-pass algorithm storing at most m elements at once, the probability of failing to report element x is, asymptotically, at least $(e^{-1+1/e})^f \approx 0.5314636^f$. Consequently, if $f = \Theta(1)$, there is a constant probability of failure, and f must be $\Omega(\lg n)$ to achieve a polynomially small probability of failure.*

5 Conclusion

The main open problem that remains is to consider the more relaxed but highly natural oblivious-adversary worst-case model, which allows randomization internally to the algorithm but assumes nothing about the input stream. We are hopeful that it is possible to achieve results similar to the stochastic model by augmenting our algorithm to randomly perturb the sizes of the rounds. The idea is that such perturbations prevent the adversary from knowing when the actual samples occur.

Acknowledgments. We thank the anonymous referees for their helpful comments and thorough review.

References

1. N. Alon, Y. Matias and M. Szegedy. "The space complexity of approximating the frequency moments", *STOC*, 1996, pp. 20–29.
2. B. Bloom. "Space/time trade-offs in hash coding with allowable queries", *Comm. ACM,* 13:7, July 1970, pp. 422–426.

3. M. Charikar, K. Chen and M. Farach-Colton. "Finding frequent items in data streams", to appear in *ICALP*, 2002.
4. S. Chaudhuri, R. Motwani and V. Narasayya. "Random sampling for histogram construction: how much is enough", In *SIGMOD*, 1998, pp. 436–447.
5. Cisco Systems. *Sampled NetFlow*, `http://www.cisco.com/univercd/cc/td/doc/product/software/ios120/120newft/120limit/120s/120s11/12s_sanf.htm`, April 2002.
6. K. Claffy, G. Miller, K. Thompson. "The nature of the beast: recent traffic measurements from an Internet backbone." In *Proc.* 8^{th} *Ann. Internet Soc. Conf.* 1998.
7. M. Datar, A. Gionis, P. Indyk and R. Motwani. "Maintaining stream statistics over sliding windows", In *SODA*, 2002, pp. 635–644.
8. N. G. Duffield and M. Grossglauser. "Trajectory sampling for direct traffic observation", In *Proc. ACM SIGCOMM*, 2000, pp. 271–282.
9. C. Estan and G. Varghese. "New directions in traffic measurement and accounting", In *Proc. ACM SIGCOMM Internet Measurement Workshop*, 2001.
10. M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani and J. Ullman. "Computing iceberg queries efficiently", *VLDB*, 1998, pp. 299–310.
11. J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan. "An approximate L^1-difference algorithm for massive data streams", In *FOCS*, 1999, pp. 501–511.
12. J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan. "Testing and Spot Checking of Data Streams", In *SODA*, 2000, pp. 165–174.
13. M. J. Fischer and S. L. Salzberg. "Finding a Majority Among N Votes: Solution to Problem 81-5", *J. Algorithms*, 3(4):362–380, 1982.
14. W. Feller. *An Introduction to Probability Theory and its Applications.* 3rd Edition, John Wiley & Sons, 1968.
15. P. Flajolet. "Approximate counting: a detailed analysis", *BIT*, 25:113–134, 1985.
16. P. Flajolet and G. N. Martin. "Probabilistic counting algorithms", *J. Computer and System Sciences*, 31, 1985, pp. 182–209.
17. P. B. Gibbons and Y. Matias. "New sampling-based summary statistics for improving approximate query answers", In *Proc. ACM SIGMOD International Conf. on Management of Data*, June 1998, pp. 331–342.
18. I. D. Graham, S. F. Donelly, S. Martin, J. Martens, and J. G. Cleary. Nonintrusive and accurate measurements of unidirectional delay and delay variation in the Internet. *Proc. 8th Annual Internet Society Conference*, 1998.
19. P. Gupta and N. Mckeown. "Packet classification on multiple fields", In *Proc. ACM SIGCOMM*, 1999, pp. 147–160.
20. P. J. Haas, J. F Naughton, S. Seshadri and L. Stokes. "Sampling-Based Estimation of the Number of Distinct Values of an Attribute", In *VLDB*, 1995, pp. 311–322.
21. P. Indyk. "Stable Distributions, Pseudorandom Generators, Embeddings and Data Stream Computations", In *FOCS*, 2000, pp. 189–197.
22. J. G. Kalbfleisch, *Probability and Statistical Inference*, Springer-Verlag, 1979.
23. R. Mahajan and S. Floyd. "Controlling High Bandwith Flows at the Congested Router", In *Proc. 9th International Conference on Network Protocols*, 2001.
24. R. Morris. "Counting large numbers of events in small registers", *Comm. ACM*, 21, 1978, pp. 840–842.
25. R. Motwani and P. Raghavan. *Randomized Algorithms*, Camb. Univ. Press, 1995.
26. J. S. Vitter. "Optimum algorithms for two random sampling problems", In *FOCS*, 1983, pp. 65–75.
27. K.-Y. Whang, B. T. Vander-Zanden, H. M. Taylor. "A Linear-Time Probabilistic Counting Algorithm for Database Applications", *ACM Trans. Database Systems* 15(2):208–229, 1990.