

A Generic Framework for Detecting Top-k Items from a Stream

ABSTRACT

We propose a generic framework for identifying the k most frequent items from a network data stream in a single pass, using a workspace much smaller than the stream itself. We observed that any small-space sketch that approximately maintains the frequencies of the items appearing in a stream can be utilized to extract the top- k items. We show theoretically how the Misra-Gries sketch [1] and the count-min sketch [2] can fit into this framework, analyze the space-complexities of both, and present experimental results on packet traces from a backbone network link. With the realistic assumption that the frequencies of the items follow a Zipfian distribution, we fully present an algorithm, based on the Misra-Gries sketch [1], that takes only $O(k^{1.5}/\epsilon)$ space and guarantees that the solution provided is ϵ -approximate. Under identical input distribution, the space-requirement of the top- k algorithm provided by Charikar *et al* [3] is logarithmic in the size of the stream, so our algorithm achieves significant space-reduction; and does not demand any apriori knowledge from the user about the size of the stream.

1. INTRODUCTION

With the drastic rise in network capacity and online activities of users in recent times, online mining of network data under memory and time constraints is becoming an increasingly challenging task. Extracting meaningful information from these data calls for the design of simple one-pass algorithms that are space and time-wise inexpensive; yet deliver results with provable accuracy. Here, we focus on the identification of *frequent features*, often known as “top- k ”, from massive network data streams. Some motivating applications for the problem are as follows:

1. As millions of users hit search engines like Google [5] and Bing [6], identification of the most popularly searched strings may necessitate application of a top- k algorithm. These popular keywords can be used for offering a suggestion service, or for drawing other conclusions about recent trends. For example, Ginsberg *et al.* [7] showed how the outbreak of an epidemic can be detected by the sudden rise of web queries including disease-related keywords like “flu” or “influenza”.

2. In a content-delivery network like Akamai [4], a network engineer might want to learn which five or ten of their clients are receiving the maximum traffic; in order to decide which customers need more mirrors.
3. Similarly, if the stream consisted of the geographic locations the HTTP requests are generated from, then we would be able to identify which countries (or regions, domains, etc) are responsible for the maximum requests. This may help us in placement of mirrors for the servers accordingly.
4. The most frequently appearing TCP/UDP port numbers in a traffic stream can give an idea of the most popular services; and hence can help a network administrator in planning resources accordingly.
5. The most popular keywords in the contents exchanged by users over the Internet can help in strategically placing advertisements. Some e-mail service providers already have started displaying suitably tailored advertisements to the mail users by mining keywords from the contents of the mails.

Formally, we are given a sequence of elements $S = (q_1, q_2, \dots, q_n)$, where each element $q_i \in \mathcal{O} = \{o_1, o_2, \dots, o_m\}$. The item o_i appears n_i times in the stream S . We assume the items are numbered such that $n_1 \geq n_2 \geq \dots \geq n_m$. Given a user-input integer $k = o(n)$, we are interested in identifying the k data items that occur most frequently in the stream; along with an estimate of the frequency of each of these k items.

The very fundamental problem of identifying the frequent items from a stream can be formulated in a number of ways:

1. **Relative Threshold:** For some user-given integer k , we are interested in the items that occur more than $\frac{n}{k+1}$ times in the stream. From a user’s point of view, this version is useful when the user has no apriori idea of the size of the stream, and hence poses the question as, for example, “Return the items that occupy more than 1% of the size of the stream”. Most of the existing literature [1, 8, 9, 10] has focused on this version of the problem.

2. **Absolute Threshold:** For some user-given integer k , we are interested in the items that occur more than k times in the stream. The user can pose this question when she has no idea of the size of the stream but has some idea of what the frequency of a typical heavy-hitter may be. Relatively fewer literature [11] have addressed this version of the problem.
3. **Top- k :** For some user-given integer k , we are interested in the (at most) k items that occur most frequently in the stream. Note that, from a user’s perspective, posing the question in this form demands the least knowledge about the stream: she needs to know neither the size of the stream, nor the frequency of a typical heavy-hitter. However, that makes the design of any algorithm [3] for this version hardest of the three.

Note that, the first two versions can be converted to each other if the size of the stream is known. However, the third one, top- k , cannot be converted to any of the first two. The reason is, we cannot find a threshold such that the frequency of the k^{th} most frequent item, n_k , can be guaranteed to lie above the threshold. In fact, if the stream consists only of instances of less than k items from \mathcal{O} , then n_k can even be zero!

This absence of a common threshold makes the top- k problem harder than the other two. With an algorithm that maintains the frequencies of the items approximately, an item with an actual frequency slightly lower than n_k may replace an item with an actual frequency slightly higher than n_k in the returned list, if the estimated frequency of the former is greater than the latter. In fact, Alon *et al* [12] showed that any randomized approximation algorithm for estimating the frequency of the most frequent item in the stream needs $\Omega(m)$ space. It is intuitively clear that identifying the top- k items, along with their frequencies, would need at least this much space. We therefore make the following relaxation on the problem:

1. As in [3], we assume the frequencies of the items follow a Zipfian distribution. We noticed, through some experiments, that for most attributes found in real-life network data (e.g., port numbers, IP addresses), a heavily skewed distribution like Zipfian is very common.
2. We attempt an ϵ -approximate version of the problem. Under the first assumption about data distribution, for some user-input $\epsilon \in (0, 1)$, our algorithm returns a set A of $l(> k)$ items from \mathcal{O} with the following guarantees:
 - Any item that occurs at least n_k times in S , is returned in A (no false negatives).
 - Any item in A occurs at least $(1 - \epsilon)n_k$ times in S .(few false positives).

We propose a general framework for the top- k problem. Our framework is based on the crucial observation that any sketch that maintains the counts of the items in a data stream approximately, can be used to identify the top- k items from the stream, provided the space allocated to the sketch meets certain guarantees. The error in estimating the frequency of an item is usually a decreasing function of the space budget - the more the space given to the sketch, less is the error. We borrow the following idea from [3]: the ϵ -approximate version of the top- k problem can be solved if the frequency estimation error by the sketch can be bounded by ϵn_k . Note that, this ensures the minimum estimated frequency of the k^{th} most frequent item (whose frequency is n_k) would always be above the maximum estimated frequency of an item with actual frequency $(1 - \epsilon)n_k$ or less, and hence the latter would never make it to top- k . Similarly, the minimum estimated frequency of an item with actual frequency $(1 + \epsilon)n_k$ or more would always be above n_k , and thus the item would qualify as top- k for sure. Whereas [3] proposes a sketching technique, called the count-sketch, we show, for two existing sketches ([1, 2]), how our framework can be applied, in general, for any sketch that maintains an approximate count of items.

1.1 Contributions

Our contributions can be summarized as follows:

- With our general framework, any sketch that approximately maintains the frequencies of the items in a data stream, can be used to identify the top- k items from a stream. Any such sketch has its own error bound; which is usually a decreasing function of the space allocated to the sketch. For a specific sketch, plugging in the error bound available from the analysis of the sketch, we can find how much space is required to use this sketch for solving the ϵ -approximate top- k problem. Further, assuming some distribution of the input frequencies, we can further express the space requirement as a function of the parameter of that distribution.
- We demonstrate how our framework can be applied for two existing sketches: the Misra-Gries sketch [1] and the count-min sketch [2]. The space-requirement of the Misra-Gries sketch turns out to be better than the count-min sketch, and also, the latter needs $\log(\frac{1}{\delta})$ pairwise independent hash functions (δ quantifies the likelihood that the guarantees offered by [2] are met). So, we present full details of an algorithm based on the Misra-Gries sketch. With this sketch and our assumption of the Zipfian distribution, the space requirement of the algorithm becomes $O(k^{1.5}/\epsilon)$. Note that, under identical conditions, the count-sketch-based algorithm of Charikar *et al* [3] takes $O(k \log \frac{n}{\delta})$ space, so our algorithm not only takes much smaller space; but also, the space requirement being a function of k and ϵ

only, demands no apriori knowledge of the size of the stream.

- We present results from our simulations on destination port numbers from anonymized packet header traces collected by CAIDA [13] over an OC48 link. We compared the performance of our small-space algorithm with a naive algorithm that stores the exact frequencies of all the destination port numbers observed in the stream, and takes as much space as it needs for that. The results confirm that our assumption of a Zipfian distribution is reasonable for the port numbers observed in real-life network traffic. For a stream of size 290 million, we could achieve false positive rates as low as 0.08% for $k = 60$, by maintaining as few as 6,000 items.

2. RELATED WORK

Most of the existing literature on the heavy-hitter problem have considered the relative threshold version of the problem [1, 2]. The problem of identifying the top- k items, being a harder one, has given rise to a relatively smaller volume of literature [3].

One of the most prominent work that attempted the top- k problem was the one by Charikar *et al* [3]. An exact version of the problem can be defined as follows: “return a set of l ($> k$) items, such that all the k items with highest frequencies are contained within the set”. As the authors of [3] pointed out, solving the top- k problem exactly for an arbitrary input distribution can be very hard. Even n_{l+1} can be very close to n_k ; and hence, for an adversarial input where all of $n_1, n_2, \dots, n_k, \dots, n_l, n_{l+1}$ are very large, it can be very difficult for the algorithm to distinguish between n_k and n_{l+1} . Thus, they addressed an approximate version of the problem which we have already defined in Section 1, and have attempted to solve here.

Charikar *et al* [3] offered a sketching technique that maintains an approximate count of the items appearing in the stream through a two-dimensional table. The presence of multiple rows ensure that a particular item maintains its footprint in multiple places; whereas the multiple columns reduce the likelihood that two different items introduce errors into each other’s frequency estimates. Each row in the table has an associated hash function that maps any element of the stream to a column in that row. The hash functions for the different rows are pairwise independent. As each element in the stream is received, it is mapped to one column in every row. The chosen cell in each column is a counter which is incremented by +1 or -1 - depending on the output of another hash function associated with the row, which maps every item to +1 or -1. These second set of hash functions are also pairwise independent. The approximate frequency of any item that has appeared in the stream can be

retrieved from this structure.

Our generic framework is based on our observation that any sketching technique that maintains an approximate count of the items appearing in a stream can be used to identify the top- k items from the stream, *as long as the error in frequency estimation is within a certain bound*. We therefore, surveyed the other sketching techniques, particularly the ones designed to address the heavy-hitter problem; and for two of them, we demonstrate how those sketching techniques can be incorporated into our framework. One of the earliest such sketches was by Misra and Gries [1]. For a sequence of elements $S = (q_1, q_2, \dots, q_n)$ and a user-input threshold $\phi \in (0, 1)$, they offered a *two-pass* deterministic algorithm to find the data items that occur more than ϕn times in the sequence. Their algorithm required $O(n \log \frac{1}{\phi})$ time and $O(\frac{1}{\phi})$ space for the sketch. A *single-pass, approximate* variant of the original algorithm provides the following approximation guarantees, for some user-input threshold ϕ and approximation error $\epsilon' < \phi$ (note that for an *online* algorithm, n is the number of elements received *so far*):

- All items whose frequencies exceed ϕn are output. There are no false negatives.
- No item with frequency less than $(\phi - \epsilon')n$ is output.
- Estimated frequencies are less than true frequencies by at most $\epsilon'n$.

The basic Misra-Gries algorithm is simple: it maintains a hashtable (of size at most $\frac{1}{\epsilon'}$) of (value, count) pairs. On receiving each item q_i , we check whether the value q_i is already in the hashtable. If it exists, we increment its count by 1; otherwise, we add the pair $(q_i, 1)$ to the hashtable. Now, if adding a new pair to the hashtable makes its size exceed $\frac{1}{\epsilon'}$, then for each of the (value, count) pairs in the hashtable, we decrement the count by one; and throw away any value whose count falls to zero after decrement. Note that this ensures at least the element which was most recently added (with a count of one) would get discarded, so the size of the hashtable, after processing all pairs, would come down to $\frac{1}{\epsilon'}$ or less. Thus, the space requirement of this algorithm is $O(\frac{1}{\epsilon'})$. Also, note that decrementing the frequencies of all the stored items, when the size of the hashtable exceeds $\frac{1}{\epsilon'}$, ensures that any existing item in the table would eventually have its estimated count reduced to zero and therefore would get discarded, if it does not occur frequently enough in the stream since it gets into the table.

The other sketching technique that we adapt here for top- k is by Cormode and Muthukrishnan [2], and is known as the count-min sketch. Like [3], the count-min sketch also maintains a two-dimensional table; and has a hash function associated with each row in the table. However, when an element in the stream is mapped onto a particular column, the counter in the cell is incremented by the count of the element

in the stream. Accordingly, the mechanism to retrieve the frequency-estimate of an item is also different from [3]. The two-dimensional table needs $O(\frac{1}{\epsilon'})$ columns and $O(\ln \frac{1}{\delta})$ rows to produce an ϵ' -approximate estimate of the frequency of an item (with probability at least $1 - \delta$). Note that, in comparison, the two-dimensional table for the count-sketch structure needs $\Omega(\frac{1}{\epsilon'^2})$ columns and $\Omega(\ln \frac{n}{\delta})$ rows to produce ϵ' -approximate estimates. Thus, the space requirements of both the Misra-Gries and the count-min sketch depend on the approximation parameters ϵ' and δ only; and not on the stream size n ; moreover, the dependency on ϵ' is $O(\frac{1}{\epsilon'})$ and not $O(\frac{1}{\epsilon'^2})$ unlike the count-sketch. These precisely were the factors that motivated us to choose these sketches for the top- k problem.

3. ALGORITHM

We have already formalized the stream model and the problem definition in Section 1. We propose a general framework to address this problem. Any algorithm in this framework would require a sketch \mathcal{A} for maintaining an estimate of the frequencies of the items that appear in the stream. The sketch \mathcal{A} can be adapted from any existing sketch that maintains a count, e.g., the Misra-Gries sketch [1] or the count-min sketch [2]. All these sketches provide some guarantee (deterministically or with high probability) on at most how much the the estimated frequency of an item, as per \mathcal{A} , can deviate from its true frequency in the stream. A data structure with the top- k items can be created from the sketch \mathcal{A} .

To guarantee that an item with frequency close to n_k but higher than n_k does not get replaced from the returned list by an item with frequency close to n_k but lower than n_k , we have to make sure that the *minimum* estimated frequency of the former is bigger enough compared to the *maximum* estimated frequency of any item of the latter.

The pseudocode of our algorithm, that meets the guarantees as mentioned in Section 1, is in Figure 1. Our algorithm takes $O(\frac{k^{1.5}}{\epsilon})$ space for maintaining the Misra-Gries sketch. The set of k most frequent items can be created from this sketch on demand.

4. ANALYSIS

The first lemma is a known result about the Misra-Gries algorithm. We mention it here for the sake of completeness.

LEMMA 4.1. *Any item $o_i \in \mathcal{O}$ with frequency n_i in the stream has at least $n_i - \frac{n}{s}$ occurrences in H at termination.*

PROOF. The total number of increments in the s counters that keep track of the counts of the different items is n . Now, every time the size of H exceeds s and some elements are discarded from it, there are $s + 1$ decrements. The total number of decrements, however, cannot be more than

Input: Data stream $S = (q_1, q_2, \dots, q_n)$; the number (k) of most frequent items to be returned; approximation parameter ϵ .

Output: A set A of l items such that

1. Any item that occurs at least n_k times in S , is returned in A .
2. Any item in A occurs at least $(1 - \epsilon)n_k$ times in S .

Algorithm:

1. Initialization
 - $H \leftarrow \Phi$
 - $/*s$ is the maximum number of distinct items stored by the algorithm*/
 - $s \leftarrow \frac{2.6k^{1.5}}{\epsilon}; l = \frac{k}{(1-\epsilon)^{2/3}}$
 2. On receiving an element x
 - if ($x \in H$) then
 - $\hat{f}_x \leftarrow \hat{f}_x + 1$
 - else
 - $/*x$ is not present in H^* /
 - insert x into H with $\hat{f}_x = 1$
 - if ($|H| > s$) then
 - for each $d \in H$
 - $\hat{f}_d \leftarrow \hat{f}_d - 1$
 - if ($\hat{f}_d = 0$) then discard d from H
 - endfor
 - endif
 - endif
 3. When asked for the frequent elements
 - create the set A with the l items with highest values of estimated frequencies from H ;
 - return A
-

Figure 1: Algorithm for identifying the k items with maximum frequencies in the stream S

the total number of increments, and hence is at most n . So the number of times H is cleaned up is at most $\frac{n}{s+1} < \frac{n}{s}$. So, for any item $o_i \in \mathcal{O}$ with frequency n_i , the number of occurrences in H at termination is at least $n_i - \frac{n}{s}$. \square

THEOREM 4.1. *With $s \geq \frac{n}{\epsilon n_k}$, a set A of k items with highest estimated frequencies as per the Misra-Gries sketch, meets the following guarantees:*

1. Any item in A occurs at least $(1 - \epsilon)n_k$ times in S .

2. Any item that occurs at least $(1 + \epsilon)n_k$ times in S , is returned in A .

PROOF. By lemma 4.1, the estimated frequency, \hat{n}_i , of an item $o_i \in \mathcal{O}$, can be at most $\frac{n}{s}$ less than its true frequency n_i in the stream. Thus, if the actual frequencies of two items differ by more than $\frac{n}{s}$, then the estimated frequency of the one with higher actual frequency will be greater than the estimated frequency of the other. By setting $\frac{n}{s} \leq \epsilon n_k$, we ensure that an item which should qualify to the top- k list can be replaced (in A) by another item only if the true frequency of the latter is at least $(1 - \epsilon)n_k$. Also, under this condition, no item with actual frequency above $(1 + \epsilon)n_k$ should be replaced (in A) by any other item whose actual frequency is less than n_k . Note that,

$$\begin{aligned} \frac{n}{s} &\leq \epsilon n_k \\ \Rightarrow s &\geq \frac{n}{\epsilon n_k} \end{aligned} \quad (1)$$

This completes the proof of the theorem. \square

Note that, when we process the stream in a *single pass*, n_k will not be known *a priori*. However, if we make an assumption about the frequency distribution of the different items, then the expression $\frac{n}{\epsilon n_k}$ can be reduced to a function of k and ϵ , thereby making the algorithm applicable for processing real-life streams in single pass. To check what assumption about the distribution is realistic for real-life network data streams, we first ran a naive algorithm on a stream of destination port numbers. The naive algorithm simply stored the exact frequencies of all the distinct port numbers in the stream. We allowed the naive algorithm to consume as much space as it needed. We then listed the top- k items based on the counts maintained by the naive algorithm, and a manual analysis revealed that an assumption of Zipfian distribution, with $n_k = \frac{c}{k^z}$, is reasonable, where c is a normalizing constant, and z lies between 1.5 and 3.

THEOREM 4.2. *If $n_k = \frac{c}{k^z}$, where $z > 1$, then maintaining $s \geq \frac{\zeta(z)k^z}{\epsilon}$ satisfies the space requirement demanded by Theorem 4.1, as well as meets the accuracy guarantees provided by Theorem 4.1, where $\zeta(z)$ is the Riemann ζ -function.*

PROOF. Note that

$$n = \sum_{i=1}^m n_i = c \sum_{i=1}^m \frac{1}{i^z} \leq c \sum_{i=1}^{\infty} \frac{1}{i^z} \leq c\zeta(z) \quad (2)$$

By Maclaurin-Cauchy test, we know for $z > 1$, the series represented by $\zeta(z)$ converges, and $\zeta(1.5) \approx 2.612$, $\zeta(2) \approx 1.645$, and so on. Also, note that

$$\begin{aligned} \frac{n}{\epsilon n_k} &= \frac{nk^z}{c\epsilon} \\ &\leq \frac{c\zeta(z)k^z}{c\epsilon} \text{ [Using inequality 2]} \\ &= \frac{\zeta(z)k^z}{\epsilon} \end{aligned} \quad (3)$$

Combining inequalities 3 and 1, we see that maintaining $s \geq \frac{\zeta(z)k^z}{\epsilon}$ implies $s \geq \frac{n}{\epsilon n_k}$, so the space demand of Theorem 4.1 is met, and the accuracy guarantees provided by Theorem 4.1 should be delivered. \square

Since $\zeta(z)$ decreases with increasing z , we used a ‘‘conservative’’ space budget assuming $z = 1.5$, thereby setting $s = \frac{2.6k^{1.5}}{\epsilon}$.

Optimization: As Charikar *et al* [3] showed, with the assumption of Zipfian distribution, we can return a set A of $l > k$ items so that *all* the top- k items are included in A . This can be ensured if $n_{l+1} < (1 - \epsilon)n_k$. Maintaining $l = \frac{k}{(1-\epsilon)^{1/z}}$ items in A is enough to ensure that there are *no false negatives*.

5. SIMULATION

We simulated our algorithm, in C++, on a test dataset with 290 million packet headers. First, we naively maintained a counter for each distinct destination port number observed in the stream; and then extracted the top k of them, the values of k being 20, 40 and 60 in different runs. In Figure 2, we plot (by direct calculations) the values of $\frac{2.6k^{1.5}}{\epsilon}$ for different values of k and ϵ to show the amount of space saved by our algorithm. The space-saving will be even more pronounced for input attributes with larger number of possible values (e.g., IP address). In Figures 3 and 4, for each of the three different values of k , we plot, from experiments, the false positive rates produced by our small-space approximation algorithm for three different values of ϵ . For a given pair (k, ϵ) , we define the *false positive rate* as the fraction of the port numbers not reported as top- k by the naive algorithm, that are (falsely) reported as top- k by the small-space algorithm. We observed that for a given value of k , as we expect, the false positive rate decreases with decreasing value of ϵ . Also, for a given value of ϵ , the false positive rate is higher for higher values of k .

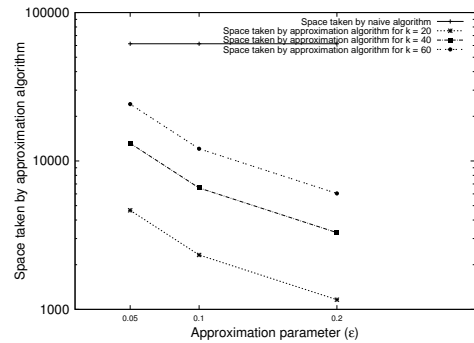


Figure 2: Space taken by the small-space algorithm for different combinations of k and ϵ

6. EXTENSION TO OTHER SKETCHES

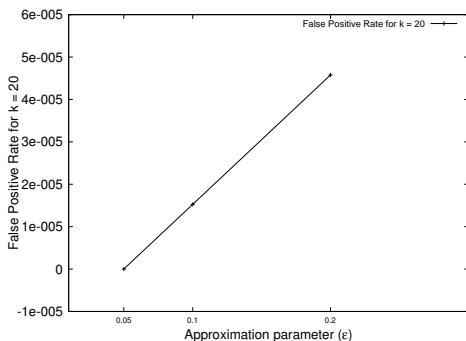


Figure 3: The false positive rates for different values of ϵ and for $k = 20$.

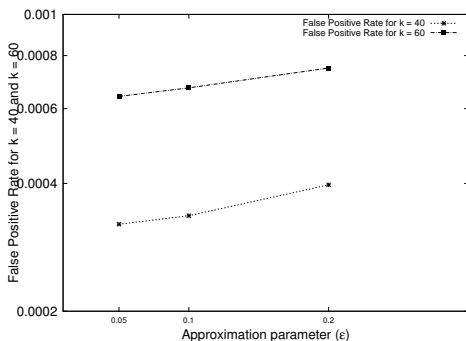


Figure 4: The false positive rates for different values of ϵ and for $k \in \{40, 60\}$.

As we have already discussed, any sketch \mathcal{A} that maintains an approximate estimate of the frequencies of the items appearing in the stream, can fit into our framework. If we use the count-min sketch [2], then the estimated frequency, \hat{n}_i , of an item $o_i \in \mathcal{O}$, meets the following guarantee with probability at least $1 - \delta$, for some user-input $\delta, \epsilon' \in (0, 1)$:

$$n_i \leq \hat{n}_i \leq n_i + \epsilon' n$$

where the count-min data structure is one with $d = \lceil \ln \frac{1}{\delta} \rceil$ rows and $w = \lceil \frac{n}{\epsilon'} \rceil$ columns. We can therefore use the count-min sketch to return a set A of k items with guarantees provided as in Theorem 4.1. However, for the space requirement of the count-min data structure, using a reasoning similar as in Theorem 4.1, we can conclude that the following criterion must be satisfied:

$$\begin{aligned} (1 - \epsilon)n_k + \epsilon' n &\leq n_k \\ \Rightarrow \frac{1}{\epsilon'} &\geq \frac{n}{\epsilon n_k} \\ \Rightarrow w &= \frac{e}{\epsilon'} \\ &\geq \frac{\epsilon n}{\epsilon n_k} \end{aligned}$$

Thus, the minimum space requirement of the count-min sketch, for identifying the top- k items, would be $\frac{\epsilon n}{\epsilon n_k} \ln \frac{1}{\delta}$. As we have shown in Theorem 4.2, $\frac{\zeta(z)k^z}{\epsilon}$ is an upper bound on $\frac{n}{\epsilon n_k}$; and hence, $\frac{e\zeta(z)k^z}{\epsilon} \ln \frac{1}{\delta}$ would be enough space for identifying the top- k items correctly using the count-min sketch. Asymptotically, this is $O(\ln \frac{1}{\delta})$ times higher than the space requirement of the Misra-Gries sketch-based top- k algorithm. It is the higher space complexity of the count-min sketch, in addition to the need of $\log(\frac{1}{\delta})$ pairwise independent hash functions, that makes the Misra-Gries algorithm a better choice for this particular problem.

7. REFERENCES

- [1] Misra, J., Gries, D.: Finding repeated elements. *Science of Computer Programming* 2(2) (1982) 143–152
- [2] Cormode, G., Muthukrishnan, S.: An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms* 55(1) (2005) 58–75
- [3] Charikar, M., Chen, K., Farach-Colton, M.: Finding frequent items in data streams. *Theoretical Computer Science* 312(1) (2004) 3–15
- [4] <http://www.akamai.com/>
- [5] <http://www.google.com/>
- [6] <http://www.bing.com/>
- [7] Ginsberg, J., Mohebbi, M.H., Patel, R.S., Brammer, L., Smolinski, M.S., Brilliant, L.: Detecting influenza epidemics using search engine query data. In: *Nature*. (19 February 2009) 1012–1014
- [8] Demaine, E.D., Lopez-Ortiz, A., Munro, J.I.: Frequency estimation of internet packet streams with limited space. In: *Proceedings of the 10th Annual European Symposium on Algorithms (ESA)*. (2002) 348–360
- [9] Manku, G.S., Motwani, R.: Approximate frequency counts over data streams. In: *Proceedings of 28th International Conference on Very Large Data Bases (VLDB)*. (2002) 346–357
- [10] Cormode, G., Muthukrishnan, S.: What’s hot and what’s not: tracking most frequent items dynamically. In: *Proceedings of the 22nd ACM SIGMOD International Conference on Management of Data / Principles of Database Systems (PODS)*. (2003) 296–306
- [11] Lahiri, B., Tirthapura, S.: Computing frequent elements using gossip. In: *Proceedings of the 15th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*. (2008) 119–130
- [12] Alon, N., Matias, Y., Szegedy, M.: The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences* 58(1) (1999) 137–147
- [13] <https://data.caida.org>