

# Finding Correlated Heavy-Hitters over Data Streams

Bibudh Lahiri\*, Srikanta Tirthapura\*

\*Department of Electrical and Computer Engineering

Iowa State University

Ames, IA, USA

{bibudh, snt}@iastate.edu

**Abstract**—We consider online mining of correlated heavy-hitters (CHH) from network data streams. Given a multi-dimensional dataset, a correlated aggregate query first filters a subset by applying a predicate along a *primary* dimension, and then computes aggregates along a *secondary* dimension of that subset data.

We consider queries of the following form: “In a stream of  $(x, y)$  tuples, on the subset  $H$  of all  $x$  values that are heavy-hitters, maintain those  $y$  values that occur frequently with the  $x$  values in  $H$ ”. This query arises naturally in situations where we need to track not only the identity of frequently occurring elements in a stream, but also additional information associated with these elements along other dimensions.

Prior work on tracking heavy-hitters has focused only on tracking the identity and frequency of heavy-hitters on a single dimensional stream, and yield little information about correlated heavy-hitters. Our online data stream algorithm is easy to implement and uses workspace which is orders of magnitude smaller than the stream itself. We present provable guarantees on the maximum error estimates, as well as experimental results, that demonstrate the space-accuracy trade-off on a large data stream of packet headers from a backbone network link.

**Keywords**-Data streams; correlation; heavy-hitters; sketches

## I. INTRODUCTION

Correlated aggregates ([1], [2], [3]) reveal interesting interactions among the different attributes of a multi-dimensional dataset. They are useful when we are interested in finding an aggregate on an attribute over a subset, where the subset can be precisely defined by another query on a different attribute of the same dataset. On a stored database, the aggregations can be performed along one dimension at a time, using multiple passes through the data. However, for streaming data, we do not have the luxury of making multiple passes over the dataset, as we get to see each data item only once. Moreover, even the subset derived by winnowing along the primary dimension can be too large to store, let alone the whole dataset.

In this work, we focus on the identification of correlated heavy-hitters (CHH) from massive data streams. We first define the notion of a heavy-hitter on a data stream. Given

a sequence of single-dimensional records  $(a_1, a_2, \dots, a_N)$  and a user-input threshold  $\phi \in (0, 1)$ , any data item that occurs more than  $\phi N$  times in the stream is termed as a  $\phi$ -heavy-hitter. We are interested in the following problem: given a data stream  $S$  of  $(x, y)$  tuples of length  $N$  ( $x$  and  $y$  will henceforth be referred to as the “primary” and the “secondary” dimensions, respectively), and two user-defined thresholds  $\phi$  and  $\psi$ , where  $0 < \phi < 1$  and  $0 < \psi < 1$ , identify all  $(d, s)$  tuples such that:

$$f_d = |\{(x, y) \in S : (x = d)\}| > \phi N$$

and

$$f_{d,s} = |\{(x, y) \in S : (x = d) \wedge (y = s)\}| > \psi f_d$$

The above aggregate can be understood as follows. The elements  $d$  are heavy-hitters in the traditional sense, on the stream formed by projecting along the primary dimension. For each heavy-hitter  $d$  along the primary dimension, there is logically a (uni-dimensional) substream  $S_d$ , consisting of all values along the secondary dimension, where the primary dimension equals  $d$ . We require the tracking of all tuples  $(d, s)$  such that  $s$  is a heavy-hitter in  $S_d$ .

### A. Applications.

Many stream mining problems on multi-dimensional streams need the CHH aggregate, and cannot be answered by independent aggregation along single dimensions. For example:

- In a stream of (server IP address, port number) tuples, identifying the heavy-hitter server IP addresses will tell us which servers are popular, and identifying frequent port numbers will tell us which applications are popular; but a network manager maybe interested in knowing which applications are popular among the heavily loaded servers. Such correlation queries can be used for network optimization, or for knowing the normal traffic pattern in a network so that an anomaly can be detected [4].
- The recommendation system of any typical online shopping site shows a buyer a list of the items frequently bought with the ones she has decided to buy. Our

The work of the authors was supported in part by grants 0834743 and 0831903 from the National Science Foundation.

algorithm can optimize the performance of such a system by parsing the transaction logs and identifying the items that were bought commonly with the frequently purchased items. If such information is stored in a cache with a small lookup time, then for most buyers, the recommendation system can save the time to perform a query on the disk-resident data.

None of the above would have been possible using the traditional heavy-hitter algorithms on streams, which only keep track of frequently occurring elements along a single dimension, and do not consider correlations across multiple dimensions.

### B. Contributions

Our contributions are as follows.

- We formalize the notion of correlated heavy-hitters (CHH), and present a small-space approximation algorithm for identifying them. Prior literature on correlated aggregates have mostly focused on the correlated sum, and these techniques are not applicable for CHH. Our summary of the two-dimensional stream is based on a nested version of the Misra-Gries sketch [5].
- We provide a provable guarantee on the approximation error. We show that there are no false negatives, and the error in the false positives is controlled. When greater memory is available, this error can be reduced. We solved a constrained optimization problem to minimize the total space taken by the algorithm to give user desired error guarantees.
- We present results from our simulations on a stream of more than 1.4 billion (50 GB trace) anonymized packet headers collected by CAIDA on an OC48 link. We compared the performance of our small-space algorithm with a slow, but exact algorithm that goes through the input data in multiple passes. Our experiments revealed that even with a space budget of a few megabytes, the average error statistics of our algorithm were very small, showing that this is a viable algorithm in practice.

One possible solution to the CHH problem would be filtering along the primary dimension using a standard heavy-hitter algorithm, and then maintaining a histogram along the secondary dimension of the filtered dataset, and then using this histogram to identify CHHs. However, our experiments with real network data (Figure 2) show that the secondary dimension can have a large number of distinct values associated with each heavy-hitter along the primary dimension. Thus, while the number of CHHs may be small, such a histogram maybe large, and the resulting algorithm would have a large space footprint.

Our algorithm exploits the fact that the data along either dimension is usually *skewed*, to maintain the heavy-hitters along the second dimension in small space. Along each

dimension, it maintains frequency estimates of mostly those values (or pairs of values) that occur frequently. For example, for every destination that sends a significant fraction of traffic on a link, we maintain mostly the sources that occur frequently along with this destination. Note that the set of heavy-hitters along the primary dimension can change as the stream elements arrive, and this influences the set of CHHs along the secondary dimension. For example, if an erstwhile heavy-hitter destination  $d$  no longer qualifies as a heavy-hitter with increase in  $N$  (and hence gets rejected from the sketch), then a source  $s$  occurring with  $d$  should also be discarded from the sketch. This interplay between different dimensions has to be handled carefully during algorithm design.

## II. RELATED WORK

In the data streaming literature, there is a significant body of work on correlated aggregates ([1], [2], [3]), as well as on the identification of heavy hitters ([6], [5], [7], [8]). See [9] for a recent overview of work on heavy-hitter identification. None of these works consider correlated heavy-hitters.

Gehrke *et al* [2] addressed correlated aggregates where the aggregate along the primary dimension was an extremum (min or max) or the average, and the aggregate along the secondary dimension was sum or count. For example, given a stream  $S$  of  $(x, y)$  tuples, their algorithm could approximately answer queries of the following form: “Return the sum of  $y$ -values from  $S$  where the corresponding  $x$ -values are more than twice the minimum of all  $x$ -values”. They maintained a technique called *adaptive histograms*, but these did not come with provable guarantees on performance.

Ananthakrishna *et al* [1] presented algorithms with provable error bounds for correlated sum and count. Given a stream  $S$  of  $(x, y)$  tuples, their algorithms could answer, approximately, prefix-range queries of the following form: “Return the sum (or count) of  $y$ -values from  $S$  where the corresponding  $x$ -values are at most  $f(AGG(x))$ ”. Here,  $AGG(x)$  is some aggregate that can be computed *exactly* on a stream using *limited* memory (e.g., min, max or average), and  $f$  is a simple function. Their solution was based on the quantile summary of [10]. With this technique, heavy-hitter queries cannot be used as the aggregate along the primary dimension since they cannot be computed on a stream using limited space.

Cormode, Tirthapura, and Xu [3] presented algorithms for maintaining the more general case of *time-decayed* correlated aggregates, where the stream elements were weighted based on the time of arrival. This work also addressed the “sum” aggregate, and the methods are not directly applicable to heavy-hitters.

The heavy-hitter literature has usually focused on the following problem. Given a sequence of items  $A = (a_1, a_2, \dots, a_N)$  and a user-input threshold  $\phi \in (0, 1)$ , find data items that occur more than  $\phi N$  times in  $A$ .

Misra and Gries [5] presented a deterministic algorithm for this problem, with space complexity  $O(\frac{1}{\phi})$ , and time complexity  $O(\log \frac{1}{\phi})$ , and query time complexity  $O(\frac{1}{\phi})$ . For exact identification of heavy-hitters, their algorithm took two passes. For approximate heavy-hitters, their algorithm used only one pass through the sequence, and had the following approximation guarantee. Assume user-input threshold  $\phi$  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  $\phi N$  are output. i.e. 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 algorithm is as follows. We maintain a data structure  $\mathcal{D}$  that contains at most  $\frac{1}{\epsilon}$  (key, count) pairs. On receiving each item  $a_i$ , we check whether the key  $a_i$  is already in  $\mathcal{D}$ . If it exists, we increment its count by 1; otherwise, we add the pair  $(a_i, 1)$  to  $\mathcal{D}$ . Now, if adding a new pair to  $\mathcal{D}$  makes its size exceed  $\frac{1}{\epsilon}$ , then for each (key, count) pair in  $\mathcal{D}$ , we decrement the count by one; and discard any key whose count falls to zero after decrement. This ensures at least the key which was most recently added (with a count of one) would get discarded, so the size of  $\mathcal{D}$ , 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, decrementing the frequencies of all the stored keys, when the size of  $\mathcal{D}$  exceeds  $\frac{1}{\epsilon}$ , ensures that any existing key in  $\mathcal{D}$  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  $\mathcal{D}$ . The data structure  $\mathcal{D}$  can be implemented using hash tables or balanced binary trees.

Our problem is different from the well-known problem of association rule mining. Following the terminology of [11], [12], if we consider the problem of mining all association rules of the form  $X \Rightarrow Y$  with *support*  $\phi$  and *confidence*  $\psi$ , where  $X$  and  $Y$  are 1-itemsets, then, *both*  $X$  and  $Y$  would have to be present in at least  $\phi$  fraction of the stream elements, which is clearly not our requirement.

### III. PROBLEM DEFINITION

The exact CHH problem, as defined in Section I, can be solved easily using a few passes, if the complete stream can be stored within main memory. However, this is infeasible since we usually have to work with main memory that is much smaller than the size of the data.

In the scenario of limited memory, the Misra-Gries algorithm [5] can be used to solve the above problem exactly in three passes through the data, as follows. We first describe a four pass algorithm. In the first two passes, heavy-hitters along the primary dimension are identified, using memory  $O(1/\phi)$ . Note that this is asymptotically the minimum possible memory requirement of any algorithm

for identifying heavy-hitters, since the size of output can be  $\Omega(\frac{1}{\phi})$ . In the next two passes, heavy-hitters along the secondary dimension are identified for each heavy-hitter along the primary dimension. This takes space  $O(\frac{1}{\psi})$  for each heavy-hitter along the primary dimension. The total space cost is  $O(\frac{1}{\phi\psi})$ , which is optimal, since the output could be  $\Omega(\frac{1}{\phi\psi})$  elements. The above algorithm can be converted into a *three* pass algorithm by combining the second and third passes.

However, we do not have the luxury of making multiple passes through the data in an online stream mining scenario. Since exact identification of heavy-hitters in a single pass is impossible using limited memory, we introduce the following approximate CHH problem.

We introduce additional approximation parameters,  $\epsilon$  and  $\epsilon'$  ( $0 < \epsilon < \phi$ ,  $0 < \epsilon' < \psi$ ), which stand for the approximation error along the primary and the secondary dimensions, respectively. Our algorithm will provide the following guarantees.

- 1) Any value  $d$  such that  $f_d > \phi N$ , will be reported as a heavy-hitter along the primary dimension.
- 2) No value  $d$  such that  $f_d < (\phi - \epsilon)N$ , will be reported as a heavy-hitter along the primary dimension.
- 3) For any value  $d$  reported above, any value  $s$  along the secondary dimension such that  $f_{d,s} > \psi f_d$  will be reported.
- 4) For any value  $d$  reported above, no value  $s$  along the secondary dimension such that  $f_{d,s} < (\psi - \epsilon')f_d - \delta N$ , where  $\delta = (\psi - \epsilon' + 1)\epsilon$ , will be reported as a CHH occurring along with  $d$ .

With this problem formulation, false positives are possible, but false negatives are not. In other words, we may incorrectly return elements that are not heavy-hitters, but whose frequency is close to the threshold required to be a heavy-hitter. But, we will never omit those elements that are indeed heavy-hitters.

### IV. ALGORITHM

We first explain the intuition behind our algorithm. We designed a data structure with two conceptual levels for extending the Misra-Gries algorithm to two dimensions, one level for each dimension. For each distinct key  $d$  along the primary dimension, instead of maintaining a (key, count) pair, we are effectively maintaining a data structure  $H$  containing 3-tuples of the form  $(d, \hat{f}_d, H_d)$ .  $\hat{f}_d$  is the estimated count of  $d$ , and  $H_d$  is an embedded Misra-Gries sketch for storing the values of the secondary attribute occurring with  $d$ . So,  $H_d$  stores its content in the form of (key, count) pairs, where the keys are values ( $s$ ) of the secondary attribute and the counts are the (estimated) number of occurrences of  $s$  (denoted as  $\hat{f}_{d,s}$ ) along with  $d$ .

For each of the two levels of this data structure, we have space-budgets ( $s_1$  and  $s_2$ ) that are derived from the user's demand of accuracy ( $\epsilon$  and  $\epsilon'$ ). We keep the space overhead for the first level within budget by periodically discarding values of the primary attribute that, recently, have not occurred frequently enough in the stream. Similarly, for each  $d$ , we keep the space overhead for  $H_d$  within budget by periodically discarding values of the secondary attribute that, recently, have not occurred frequently enough in the substream induced by  $d$ . At each level, elements need to be discarded to save space, and this is the reason for the inaccuracy in frequency estimates. The smaller is the space budget, more often we have to discard elements, and greater is the inaccuracy.

In our case, a pair  $(d, s)$  could incur loss in its frequency estimate because of two reasons: first,  $d$  (and hence  $H_d$ ) can get discarded from  $H$  because of not being frequent in  $S$ ; second, the pair  $(d, s)$  itself can get discarded from  $H_d$  because of not being frequent in the substream induced by  $d$ , even if  $d$  is not discarded from  $H$  (because  $d$  occurs frequently enough in  $S$ , along with other values of the secondary attribute). In the correctness proof (Section V, we present a careful analysis of the errors introduced in the above process.

We present the pseudo-code for this algorithm in Figure 1, followed by correctness proofs (Section V), theoretical analysis (Section VI), and experimental results (Section VII).

## V. CORRECTNESS

*Lemma 5.1:*  $\frac{1}{s_1} \leq \epsilon$ , and  $\frac{1}{s_2} \leq \epsilon'$  under all conditions.

*Proof:* At the initialization phase of the algorithm in Figure 1, if  $\epsilon < \frac{\phi\epsilon'}{2}$ , then we set  $s_1$  to  $\frac{1}{\epsilon}$ . On the other hand, if  $\epsilon \geq \frac{\phi\epsilon'}{2}$  (which implies  $\frac{2}{\phi\epsilon'} \geq \frac{1}{\epsilon}$ ), then we set  $s_1$  to  $\frac{2}{\phi\epsilon'}$ . Either way,  $\frac{1}{s_1} \leq \epsilon$ . Also, under all conditions,  $s_2 = \frac{1}{\epsilon' - \frac{1}{s_1\phi}} \geq \frac{1}{\epsilon'} \Rightarrow \frac{1}{s_2} \leq \epsilon'$ . ■

*Lemma 5.2:* Any value  $d$  of the primary attribute with frequency  $f_d$  has at least  $f_d - \frac{N}{s_1}$  occurrences in  $H$  at termination.

*Proof:* The total number of increments in the  $s_1$  counters that keep track of the counts of the different values of the primary attribute is  $N$ . Now, every time the size of  $H$  exceeds  $s_1$  and some elements are discarded from it, there are  $s_1 + 1$  decrements. The total number of decrements, however, cannot be more than the total number of increments, and hence is at most  $N$ . So the number of times  $H$  is cleaned up is at most  $\frac{N}{s_1 + 1} < \frac{N}{s_1}$ . So, for any value  $d$  of the primary attribute with frequency  $f_d$ , the number of occurrences in  $H$  at termination is at least  $f_d - \frac{N}{s_1}$ . ■

*Lemma 5.3:* Any value  $d$  of the primary attribute with frequency  $f_d$  has at least  $f_d - \epsilon N$  occurrences in  $H$  at termination.

*Proof:* Follows by combining Lemma 5.2 with Lemma 5.1. ■

---

**Input:** Data stream  $S = \{(x, y)\}$ , frequency thresholds  $\phi$  and  $\psi$ , approximation errors  $\epsilon$  and  $\epsilon'$ .  $N$  is the number of stream elements observed so far.

**Algorithm:**

1. Initialization

$$H \leftarrow \Phi; s_1 \leftarrow \max\left(\frac{1}{\epsilon}, \frac{2}{\phi\epsilon'}\right); s_2 \leftarrow \frac{1}{\epsilon' - \frac{1}{s_1\phi}}$$

2. On receiving a new tuple  $(x, y)$

if  $(x \in H)$  then

$$\hat{f}_x \leftarrow \hat{f}_x + 1$$

if  $(y \in H_x)$  then

/\*Both  $x$  and  $y$  are present\*/

increment  $\hat{f}_{x,y}$  in  $H_x$  by 1

else

/\* $x$  is present,  $y$  is not\*/

$H_x.put(y, 1)$

if  $(|H_x| > s_2)$  then

for each  $(s, \hat{f}_{d,s}) \in H_x$

$$\hat{f}_{d,s} \leftarrow \hat{f}_{d,s} - 1$$

if  $(\hat{f}_{d,s} = 0)$  then

discard  $(s, \hat{f}_{d,s})$  from  $H_x$

else

/\*None of  $x$  and  $y$  is present\*/

$H_x \leftarrow \Phi; H_x.put(y, 1); \hat{f}_x \leftarrow 1$

if  $(|H| > s_1)$  then

for each  $d \in H$

$$\hat{f}_d \leftarrow \hat{f}_d - 1$$

pick up a random element  $(s, \hat{f}_{d,s})$  from  $H_d$  and decrement  $\hat{f}_{d,s}$  by 1;

if  $(\hat{f}_{d,s} = 0)$  then

discard  $(s, \hat{f}_{d,s})$  from  $H_d$

if  $(\hat{f}_d = 0)$  then

discard  $d$  from  $H$

3. When asked for the frequent elements

for each  $d \in H$

if  $\hat{f}_d \geq (\phi - \epsilon)N$  then

report  $d$  as a frequent value of the primary attribute;

for each  $(s, \hat{f}_{d,s}) \in H_d$

if  $\hat{f}_{d,s} \geq (\psi - \epsilon')\hat{f}_d - \epsilon N$  then

output  $s$  as a CHH occurring with  $d$ ;

---

Figure 1. Algorithm for identifying the frequently occurring values of the primary attribute and the corresponding CHHs.  $s_1$  is the maximum number of stored distinct values of the primary attribute. For each stored value of the primary attribute,  $s_2$  is the corresponding maximum number of stored values of secondary attribute.

*Theorem 5.1:* Any value  $d$  of the primary attribute with frequency  $f_d > \phi N$  is reported by the algorithm as a frequent element.

*Proof:* From Lemma 5.3, the number of occurrences of a value  $d$  of the primary attribute, with frequency  $f_d > \phi N$ , in  $H$  at termination is at least  $f_d - \epsilon N > (\phi - \epsilon)N$ . Hence the algorithm in Figure 1 would report it as a frequently occurring value of the primary attribute. ■

*Theorem 5.2:* No value  $d$  of the primary attribute with frequency  $f_d < (\phi - \epsilon)N$  is reported by the algorithm as a frequent value.

*Proof:* The claim follows directly from the algorithm, and the fact that the number of times any value of the primary attribute occurs in  $H$  at termination is at most its true frequency. ■

*Lemma 5.4:* For a given value  $d$  of the primary attribute, any value  $s$  of the secondary attribute that occurs with  $d$  and has frequency  $f_{d,s}$  has at least  $f_{d,s} - \frac{f_d}{s_2} - \frac{N}{s_1}$  occurrences in  $H_d$  at termination.

*Proof:* The estimated count  $\hat{f}_{d,s}$  of the value  $s$  of the secondary attribute (in  $H_d$ ) can be less than the true count  $f_{d,s}$  because of decrements at two levels:

- 1) Decrements in  $\hat{f}_{d,s}$  that take place with decrements in  $\hat{f}_d$ , when size of  $H$  exceeds  $s_1$ .
- 2) Decrements in  $\hat{f}_{d,s}$  that take place when size of  $H_d$  exceeds  $s_2$ .

We calculate the loss in  $\hat{f}_{d,s}$  because of these two levels separately. For level 1, as we have proven in Lemma 5.2, there can be at most  $\frac{N}{s_1}$  decrements in  $\hat{f}_d$ , and each time there is a decrement in  $\hat{f}_d$ , the count  $\hat{f}_{d,s}$  of some (arbitrarily chosen)  $s \in H_d$  is decremented by 1. The maximum loss in  $\hat{f}_{d,s}$ , because of the decrements in  $\hat{f}_d$  that take place with decrements in  $\hat{f}_d$ , can thus be at most  $\frac{N}{s_1}$ .

For level 2, each time the size of  $H_d$  exceeds  $s_2$ , there are  $s_2 + 1$  decrements. But the total number of decrements cannot be more than  $f_d$ , because the total frequency of all the values of the secondary attribute associated with a particular value  $d$  of the primary attribute cannot be more than  $f_d$ . So, the total number of times  $H_d$  is reduced to size  $s_2$ , by discarding one or more elements from it, is at most  $\frac{f_d}{s_2+1} < \frac{f_d}{s_2}$ . During each discard operation, a value  $s$  of the secondary attribute can have a loss of count of at most 1. So, the maximum decrement in the estimated count of a value  $s$  of the secondary attribute, due to level 2 decrements, is at most  $\frac{f_d}{s_2}$ . ■

*Lemma 5.5:* For a given value  $d$  of the primary attribute, any value  $s$  of the secondary attribute that occurs with  $d$  and has frequency  $f_{d,s}$  has at least  $f_{d,s} - \epsilon' f_d - \epsilon N$  occurrences in  $H_d$  at termination.

*Proof:* The claim follows by combining Lemma 5.4 with Lemma 5.1. ■

*Theorem 5.3:* For any (heavy-hitter) value  $d$  of the primary attribute, any value  $s$  of the secondary attribute that

occurs with  $d$  such that  $f_{d,s} > \psi f_d$  will be identified by the algorithm as a CHH occurring alongwith  $d$ .

*Proof:* From Lemma 5.5, the estimated frequency of  $s$  in  $H_d$ ,  $\hat{f}_{d,s}$ , at termination is at least

$$\begin{aligned} f_{d,s} - \epsilon' f_d - \epsilon N &> \psi f_d - \epsilon' f_d - \epsilon N \\ &= (\psi - \epsilon') f_d - \epsilon N \\ &\geq (\psi - \epsilon') (\hat{f}_d) - \epsilon N \end{aligned}$$

$(\psi - \epsilon') (\hat{f}_d) - \epsilon N$  is the threshold based on which a value of the secondary attribute is reported as frequent, as per the algorithm in Figure 1. ■

*Theorem 5.4:* For any (heavy-hitter) value  $d$  of the primary attribute, no value  $s$  of the secondary attribute that occurs with  $d$  such that  $f_{d,s} < (\psi - \epsilon') f_d - \delta N$ , where  $\delta = (\psi - \epsilon' + 1)\epsilon$ , will be reported as a CHH occurring alongwith  $d$ .

*Proof:* As per the algorithm, a value  $s$  of the secondary attribute is not reported as frequent if the estimated frequency of the pair  $(d, s)$  in  $H_d$  is less than  $(\psi - \epsilon') \hat{f}_d - \epsilon N$ . Now,

$$\begin{aligned} (\psi - \epsilon') \hat{f}_d - \epsilon N &\geq (\psi - \epsilon') (f_d - \epsilon N) - \epsilon N \\ &= (\psi - \epsilon') f_d - \delta N \end{aligned}$$

since  $(\psi - \epsilon' + 1)\epsilon = \delta$ . So, any value  $s$  of the secondary attribute that occurs with  $d$  such that  $f_{d,s} < (\psi - \epsilon') f_d - \delta N$ , will have its estimated frequency in  $H_d$  less than the threshold  $(\psi - \epsilon') \hat{f}_d - \epsilon N$ . ■

## VI. ANALYSIS

The  $\epsilon$ -approximate Misra-Gries algorithm [5], at any point of time, maintains at most  $O(\frac{1}{\epsilon})$  counters. In our algorithm, we maintain at most  $s_2$  counters for each of the (at most)  $s_1$  distinct values of the primary attribute in  $H$ . So the size of our sketch is at most  $s_1 + s_1 s_2 = O(s_1 s_2)$ . Next, we prove that the space allocation strategy at the two levels in our algorithm, as defined during the initialization phase in Figure 1, ensures that the total space taken is a minimum under the given precision constraints. Suppose, the precision requirements from the user are the following:

- 1) The estimated frequency of a frequent value  $d$  of the primary attribute should be at most  $\epsilon N$  less than its true frequency  $f_d$ , i.e.,  $f_d - \hat{f}_d \leq \epsilon N$ .
- 2) For any frequent value  $d$  of the primary attribute with frequency  $f_d$ , and a CHH  $s$  that occurs with  $d$ , the estimated frequency of the pair  $(d, s)$  should be at most  $\epsilon' f_d$  less than its true frequency  $f_{d,s}$ , i.e.,  $f_{d,s} - \hat{f}_{d,s} \leq \epsilon' f_d$ .

The second requirement from the user offers a tighter bound on the absolute error in measuring  $f_{d,s}$  than Lemma 5.5 in Section V offers. In fact, Lemma 5.5 was derived only to prove the subsequent claims in Section V, but this precision requirement from the user is more meaningful. With these

precision requirements from the user, the space allocations at the two levels,  $s_1$  and  $s_2$ , should meet the following criteria:

*Lemma 6.1:* The precision requirements from the user would be satisfied if the space allocations for the two levels,  $s_1$  and  $s_2$ , satisfy the following two constrains

$$\frac{1}{s_1} \leq \epsilon \quad (1)$$

$$\frac{1}{\phi s_1} + \frac{1}{s_2} \leq \epsilon' \quad (2)$$

*Proof:* For Inequality 1, we proved in Lemma 5.2 that the maximum difference between the actual and estimated frequencies of a value  $d$  of the primary attribute is at most  $\frac{N}{s_1}$ . So, a sufficient condition that would meet the first precision requirement from the user, as mentioned above, would be  $\frac{1}{s_1} \leq \epsilon$ , which is in fact met by our algorithm (as we have shown in Lemma 5.1).

For Inequality 2, we proved in Lemma 5.4 that the difference between the actual and estimated frequencies of a value pair  $(d, s)$  is at most  $\frac{f_d}{s_2} + \frac{N}{s_1}$ . Now, for a frequent value  $d$  of the primary attribute,  $f_d \geq \phi N$  (which implies  $N \leq \frac{f_d}{\phi}$ ), so the difference between the actual and estimated frequencies of  $(d, s)$  is

$$\begin{aligned} f_{d,s} - \hat{f}_{d,s} &\leq \frac{f_d}{s_2} + \frac{N}{s_1} \\ &\leq \frac{f_d}{s_2} + \frac{f_d}{\phi s_1} \text{ [Since } d \text{ is a frequent value]} \\ &= \left( \frac{1}{\phi s_1} + \frac{1}{s_2} \right) f_d \end{aligned} \quad (3)$$

A sufficient condition that would meet the second precision requirement from the user, as mentioned above, would be  $\frac{1}{\phi s_1} + \frac{1}{s_2} \leq \epsilon'$ . ■

*Theorem 6.1:* Under the constraints given in Lemma 6.1, the space allocation strategy of the algorithm in Figure 1, i.e.,  $s_1 = \max(\frac{1}{\epsilon}, \frac{2}{\phi \epsilon'})$  and  $s_2 = \frac{1}{\epsilon' - \frac{1}{s_1 \phi}}$ , ensures that the total space taken by the algorithm,  $s_1 s_2$ , is a minimum.

*Proof:* Let  $\sigma_i = \frac{1}{s_i}$  for  $i = 1, 2$ . Then the equivalent claim would be this:  $\sigma_1 \sigma_2$  is maximized under the following constraints:

$$\sigma_1 \leq \epsilon \quad (4)$$

$$\frac{\sigma_1}{\phi} + \sigma_2 \leq \epsilon' \quad (5)$$

Since the objective function  $\sigma_1 \sigma_2$  increases monotonically (in fact, linearly) with  $\sigma_2$ , any point  $(\sigma_1', \sigma_2')$  that maximizes  $\sigma_1 \sigma_2$  should lie on the line  $\frac{\sigma_1}{\phi} + \sigma_2 = \epsilon'$ . So, for any optimal solution point,  $\sigma_2 = \epsilon' - \frac{\sigma_1}{\phi}$ , so we can write the objective function as a function of  $\sigma_1$  alone, i.e.,

$$f(\sigma_1) = \sigma_1 \sigma_2 = \sigma_1 \left( \epsilon' - \frac{\sigma_1}{\phi} \right) \quad (6)$$

What remains now is to show that  $f(\sigma_1)$  in Equation 6 is maximized by the algorithm under the constraint in Inequality 4.

Differentiating  $f(\sigma_1)$  with respect to  $\sigma_1$  and equating to 0 yields  $\sigma_1 = \frac{\phi \epsilon'}{2}$ . We also see  $f''(\sigma_1) = -\frac{2}{\phi} < 0$ , so  $f(\sigma_1)$  reaches a maximum at  $\sigma_1 = \frac{\phi \epsilon'}{2}$ . Since  $f(\sigma_1)$  reaches maximum only at a single point, we can conclude that  $f(\sigma_1)$  increases monotonically for  $\sigma_1 \in [0, \frac{\phi \epsilon'}{2}]$  and it decreases monotonically for  $\sigma_1 \in [\frac{\phi \epsilon'}{2}, \epsilon]$ . Hence, if  $\epsilon < \frac{\phi \epsilon'}{2}$ ,  $f(\sigma_1)$  is maximum at  $\sigma_1 = \epsilon$ . When  $\epsilon < \frac{\phi \epsilon'}{2}$ , our algorithm sets  $s_1$  to  $\frac{1}{\epsilon}$ , which indeed satisfies  $\sigma_1 = \epsilon$ .

On the other hand, when  $\epsilon \geq \frac{\phi \epsilon'}{2}$ , then  $f(\sigma_1)$  is maximum at  $\sigma_1 = \frac{\phi \epsilon'}{2}$ . Our algorithm sets  $s_1$  to  $\frac{2}{\phi \epsilon'}$ , which indeed satisfies  $\sigma_1 = \frac{\phi \epsilon'}{2}$ . ■

*Theorem 6.2:* The total space taken by the algorithm is  $O\left(\frac{\max(\frac{1}{\epsilon}, \frac{2}{\phi \epsilon'})}{\epsilon' - \min(\frac{\epsilon}{\phi}, \frac{\epsilon'}{2})}\right)$ .

*Proof:*  $s_1 = \max(\frac{1}{\epsilon}, \frac{2}{\phi \epsilon'}) \Rightarrow \frac{1}{s_1} = \min(\epsilon, \frac{\phi \epsilon'}{2})$ . This makes the total space complexity  $O(s_1 s_2) = O\left(s_1 \cdot \left(\frac{1}{\epsilon' - \frac{1}{s_1 \phi}}\right)\right) = O\left(\frac{\max(\frac{1}{\epsilon}, \frac{2}{\phi \epsilon'})}{\epsilon' - \min(\frac{\epsilon}{\phi}, \frac{\epsilon'}{2})}\right)$ . When  $\epsilon < \frac{\phi \epsilon'}{2}$ ,  $s_1 = \frac{1}{\epsilon}$  and  $s_2 = \frac{1}{\epsilon' - \frac{\epsilon}{\phi}}$ , and the total space taken is  $s_1 s_2 = \frac{1}{\epsilon(\epsilon' - \frac{\epsilon}{\phi})}$ . On the other hand, when  $\epsilon > \frac{\phi \epsilon'}{2}$ ,  $s_1 = \frac{2}{\phi \epsilon'}$  and  $s_2 = \frac{1}{\epsilon' - (\frac{\phi \epsilon'}{2})(\frac{1}{\phi})} = \frac{2}{\epsilon'}$ , and the total space taken is  $s_1 s_2 = \frac{4}{\phi \epsilon'^2}$ . ■

## VII. SIMULATION

The goal of the simulation was threefold: first, to learn about typical frequency distributions in real two-dimensional network traffic streams; second, to illustrate the reduction in space and time cost achievable by the small-space algorithm in practice; and finally, to demonstrate how the space budget (and hence, the allocated memory) influences the accuracy of our algorithm in practice.

For the first objective, we ran a naive algorithm on a smaller dataset of 248 million (destination IP, source IP) tuples, where all the distinct destination IPs were stored, and for each distinct destination IP, all the distinct source IPs were stored. We identified (exactly) the frequent values along both the dimensions for  $\phi = 0.001$  and  $\psi = 0.001$ . Only 43 of the 1.2 million distinct destination IPs were reported as heavy-hitters. For the secondary dimension, we ranked the heavy-hitter destination IPs based on the number of distinct source IPs they co-occurred with, and the number of distinct source IPs for the top eight are shown in Figure 2. All these heavy-hitter destination IPs co-occurred with 9,000-18,000 *distinct* source IPs, whereas, for all of them, the number of co-occurring *heavy-hitter* source IPs was in the range 20-200 (note that the Y-axis in Figure 2 is in log scale). This shows that the distribution

of the primary attribute values, as well as that of the secondary attribute values for a given value of the primary attribute, are very skewed, and hence call for the design of small-space approximation algorithms like ours.

The second objective was accomplished by comparing the space and time costs of the naive algorithm as above (on the same dataset), with those of the small-space algorithm, run with  $s_1 = 3000$  and  $s_2 = 2000$  (Figure 3). We defined the space cost as the distinct number of (dstIP, srcIP) tuples stored ( $\sum_d |H_d|$ ), which is 34 times higher for the naive algorithm compared to the small-space one. Also, the naive algorithm took more than twice as much time to run the small-space one.

For the third objective, we tested the small-space algorithm on a dataset with 1.4 billion (destination IP, source IP) tuples, with different values of  $s_1$  and  $s_2$ . To test the accuracy of our small-space algorithm, we derived the “ground truth”, i.e., a list of the *actual* heavy-hitters along both the dimensions along with their *exact* frequencies, by employing a four-pass variant of the Misra-Gries algorithm (as discussed in Section III).

We define the error statistic in estimating the frequency of a heavy-hitter value  $d$  of the primary attribute as  $\frac{f_d - \hat{f}_d}{N}$ , and in Figure 4, for each value of  $s_1$ , we plot the maximum and the average of this error statistic over all the heavy-hitter values of the primary attribute. We observed that both the maximum and the average fell sharply as  $s_1$  increased. Even by using a space budget ( $s_1$ ) as low as 1000, the maximum error statistic was only 0.09%.

The graphs in Figure 5 show the results of running our small-space algorithm with different values of  $s_1$  as well as  $s_2$ . We define the error statistic in estimating the frequency of a CHH source IP  $s$  (that occurs along with a heavy-hitter destination IP  $d$ ) as  $\frac{f_{d,s} - \hat{f}_{d,s}}{f_d}$ , and for each combination of  $s_1$  and  $s_2$ , we plot the theoretical maximum, the experimental maximum and the average of this error statistic over all CHH source IPs. Here also, we observed that both the maximum and the average fall sharply as  $s_1$  increases. However, for a fixed value of  $s_1$ , as we increased the value of  $s_2$ , the maximum did not change at all, and the average did not reduce too much. The possible reason is the number of CHH source IPs being very low compared to the number of distinct source IPs occurring with a heavy-hitter destination IP, as we have pointed out in Figure 2. However, this is good because it implies that in practice, setting  $s_2$  as low as  $\frac{1}{\psi}$  should be good enough.

## REFERENCES

[1] R. Ananthakrishna, A. Das, J. Gehrke, F. Korn, S. Muthukrishnan, and D. Srivastava, “Efficient approximation of corre-

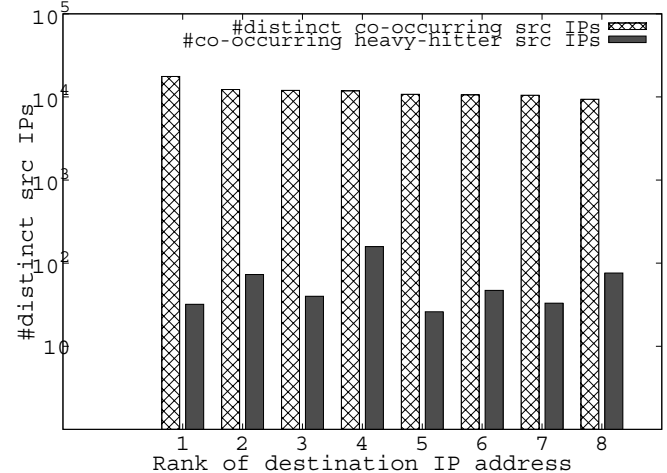


Figure 2. On the X-axis are the ranks of the eight (heavy-hitter) destination IPs, that co-appear with maximum number of distinct source IPs. For each destination IP, the Y-axis shows 1) the number of distinct source IPs co-occurring with it, 2) the number of heavy-hitter destination IPs co-appearing with it.

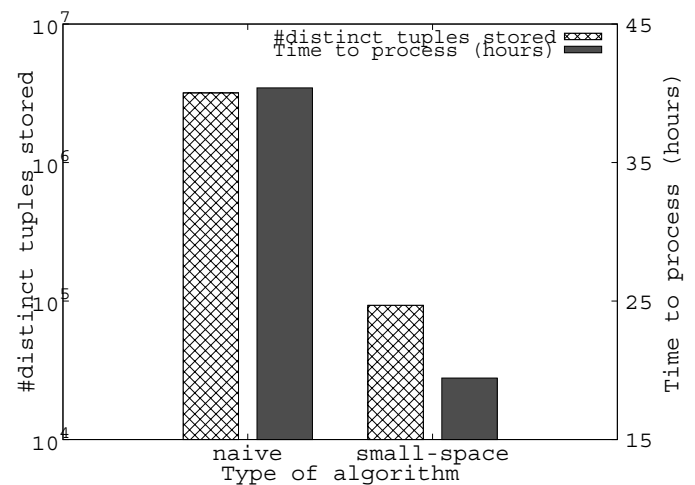


Figure 3. Comparison of space (left Y-axis) and time (right Y-axis) costs of the naive and the small-space algorithms. The time is the number of hours to process the 248 million records. Note that the left Y-axis is logarithmic, the right Y-axis is linear.

lated sums on data streams,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 3, pp. 569–572, 2003.

[2] J. Gehrke, F. Korn, and D. Srivastava, “On computing correlated aggregates over continual data streams,” in *Proceedings of the 20th ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2001, pp. 13–24.

[3] G. Cormode, S. Tirthapura, and B. Xu, “Time-decayed correlated aggregates over data streams,” in *Proceedings of the SIAM International Conference on Data Mining (SDM)*, 2009, pp. 269–280.

[4] R. E. Cullingford, “Correlation and collaboration in anomaly

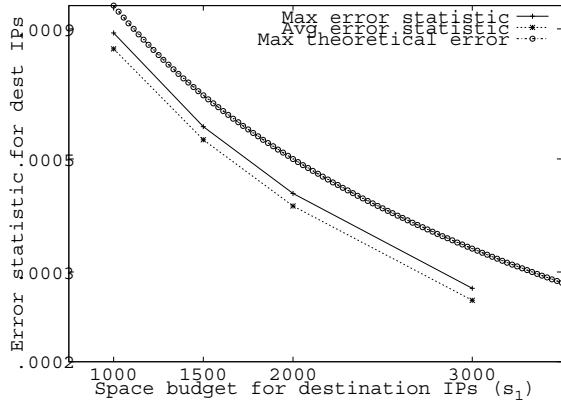


Figure 4. Error statistic in estimating the frequencies of the heavy-hitter destination IPs. The graph shows the theoretical maximum ( $\frac{1}{s_1}$ ), the experimental maximum and the experimental average.

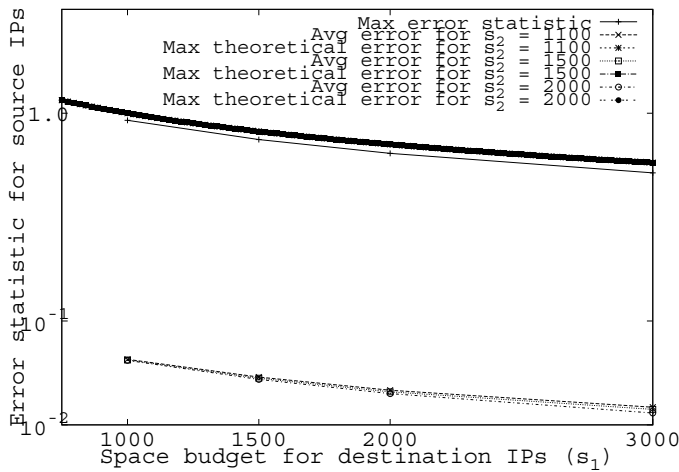


Figure 5. Error statistic in estimating the frequencies of the CHH source IPs, for  $s_2 = 1100, 1500$  and  $2000$ . The graph shows the theoretical maxima ( $\frac{1}{\phi s_1} + \frac{1}{s_2}$ ), the experimental maxima and the experimental average.

- [9] G. Cormode and M. Hadjieleftheriou, "Finding the frequent items in streams of data," *Commun. ACM*, vol. 52, no. 10, pp. 97–105, 2009.
- [10] M. Greenwald and S. Khanna, "Space-efficient online computation of quantile summaries," in *Proceedings of the 20th ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2001, pp. 58–66.
- [11] R. Agrawal, T. Imielinski, and A. N. Swami, "Mining association rules between sets of items in large databases," in *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 1993, pp. 207–216.
- [12] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in *Proceedings of 20th International Conference on Very Large Data Bases (VLDB)*, 1994, pp. 487–499.

detection," in *Cybersecurity Applications & Technology Conference For Homeland Security (CATCH)*, 2009, pp. 251–254.

- [5] J. Misra and D. Gries, "Finding repeated elements," *Science of Computer Programming*, vol. 2, no. 2, pp. 143–152, 1982.
- [6] G. S. Manku and R. Motwani, "Approximate frequency counts over data streams," in *Proceedings of 28th International Conference on Very Large Data Bases (VLDB)*, 2002, pp. 346–357.
- [7] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," *Theoretical Computer Science*, vol. 312, no. 1, pp. 3–15, 2004.
- [8] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.