Finding Heavy Distinct Hitters in Data Streams

Thomas Locher
IBM Research – Zurich
thl@zurich.ibm.com

ABSTRACT

A simple indicator for an anomaly in a network is a rapid increase in the total number of distinct network connections. While it is fairly easy to maintain an accurate estimate of the current total number of distinct connections using streaming algorithms that exhibit both a low space and computational complexity, identifying the network entities that are involved in the largest number of distinct connections efficiently is considerably harder. In this paper, we study the problem of finding all entities whose number of distinct (outgoing or incoming) network connections is at least a specific fraction of the total number of distinct connections. These entities are referred to as heavy distinct hitters. Since this problem is hard in general, we focus on randomized approximation techniques and propose a sampling-based and a sketch-based streaming algorithm. Both algorithms output a list of the potential heavy distinct hitters including the estimated counts of the corresponding number of distinct connections. We prove that, depending on the required level of accuracy of the output list, the space complexities of the presented algorithms are asymptotically optimal up to small logarithmic factors. Additionally, the algorithms are evaluated and compared using real network data in order to determine their usefulness in practice.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*computations on discrete structures*

General Terms

Algorithms, Theory

Keywords

Network Monitoring, Anomaly Detection, Streaming Algorithms, Heavy Distinct Hitter, Space Complexity

1. INTRODUCTION

Today's networks carry more and more data at increasing bitrates, which makes it progressively harder to monitor them and react to exceptional or undesirable situations in a timely fashion. An exceptional situation may be, e.g., some sort of attack on a machine in the network, a worm that propagates inside the network, or a suspicious activity that may precede an attack, such as a port scan. In particular the timeliness of detection is crucial in order to contain such incidents and limit the caused damage. In practice, intrusion prevention and detection systems such as Snort[1] are deployed, which typically store information about each network flow. Due to the increasing amount of traffic data, such simple approaches become insufficient as it becomes impractical or even infeasible to constantly sift through massive log files to spot unusual activity. Thus, there is a growing need for stream processing techniques that require little space and processing cost.

A simple and straightforward indicator for an anomaly is a substantial change in the number of distinct network connections. It is well known that this metric can be approximated accurately using little space. For example, during the spreading phase of a worm, infected machines try to establish connections to as many other entities in the network as possible. The opposite situation occurs in the case of a distributed denial of service (DDoS) attack, where many distinct entities target one or a small number of specific destinations. Note that, while such an attack often results in a significant increase in network traffic volume, there are certain DDoS attacks, e.g., *TCP SYN flooding attacks*, that do not necessarily cause a noticeable increase in bandwidth usage. In both scenarios, monitoring the number of distinct network connections allows a network administrator to quickly detect the anomaly. However, this simple approach has a major shortcoming: If an anomaly is detected, the entities that provoked it or, in case of an attack, that are targeted are not revealed, implying that log data would still have to be parsed in order to identify them. Thus, the main question is whether and how these entities can be found efficiently. Naturally, one would also like to know the number of distinct (incoming or outgoing) connections for each of those entities to quantify the severity of the situation.

In this paper, we study the following general problem, which encompasses the network security problems outlined above. Given a data stream, we consider two features of each flow, such as the source and the destination IP address. In other words, we abstract away all other features and as-

[1]See http://www.snort.org/.

sume that each flow consists of a pair of features, referred to as *element* and *value*. An element that occurs with a large number of *distinct* values in the data stream is called a *heavy distinct hitter*. The goal is to find the heavy distinct hitters and, for each heavy distinct hitter, the corresponding number of distinct values, using as little space as possible. The precise definition of the model and the considered problem is given in the following section.

It is easy to see that by using the IP source address as the element and the IP destination address as the value, each heavy distinct hitter may correspond to an infected machine that is spreading a worm, and, by using the reverse assignment, the heavy distinct hitters may be the victims of a DDoS attack. As many other important network monitoring problems can potentially be mapped to this general problem, space-efficient solutions could be highly useful for numerous practical applications. Moreover, given the connection to the fundamental problem of computing the number of distinct items for some feature, the considered problem is also interesting from a theoretical perspective. Thus, it comes as no surprise that many variations of this problem have already been studied. The most important related work is dicussed in Section 4.

As we will see, finding heavy distinct hitters is hard in general, and we therefore have to settle for randomized approximation algorithms. Section 3 presents two simple and parallelizable approximation algorithms for this problem, including their analyses. The results in this section are the main contributions of this paper as they show that there are indeed efficient solutions for the given model. Furthermore, we prove that the achieved space bounds are close to optimal, depending on the required accuracy of the approximation. Additionally, the effectiveness of the proposed algorithms is validated using traces of real network data.

2. MODEL

A data stream $\mathcal{S}$ is modeled as a sequence of n element-value pairs $(e_1, v_1), \ldots, (e_n, v_n)$. The elements and values are taken from the domains E and V, respectively. Let $m \leq n$ denote the number of *distinct* element-value pairs, where two pairs (e_i, v_i) and (e_j, v_j) are considered distinct if $e_i \neq e_j$ or $v_i \neq v_j$ (or both). Furthermore, we define that $E_{\mathcal{S}} \subseteq E$ is the set of all elements that occur as part of an element-value pair in the stream $\mathcal{S}$. The cardinality of this set, which is upper bounded by m, is denoted by $\ell := |E_{\mathcal{S}}|$.

Simply speaking, an element is called a *heavy distinct hitter* if its contribution to the total number m of distinct pairs is large.[2] The contribution of an element is quantified by means of a function $w_{\mathcal{S}} : E \to \mathbb{N}$ mapping each element e to the number of distinct values that form an element-value pair with e in $\mathcal{S}$. Formally, the function $w_{\mathcal{S}}$ is defined as $w_{\mathcal{S}}(e) := |\{v \in V \mid (e, v) \in \mathcal{S}\}|$. We say that $w_{\mathcal{S}}(e)$ is the *weight* of element e in stream $\mathcal{S}$. Note that this definition implies that $\sum_{e \in E_{\mathcal{S}}} w_{\mathcal{S}}(e) = m$. Since we always consider a specific stream $\mathcal{S}$, we simply write $w(e)$ instead of $w_{\mathcal{S}}(e)$ in the following. It remains to specify what constitutes a sufficiently large weight: An element e is a heavy distinct hitter if its weight $w(e)$ is at least a certain threshold value T, i.e., the parameter T establishes the boundary between heavy distinct hitters and the remaining elements.

Given a specific threshold T, the general goal is to compute the set $\Pi := \{(e, w(e)) \mid w(e) \geq T\}$. As argued in the previous section, it is desirable to use algorithms that require as little space as possible. The *space complexity* of an algorithm is defined as the number of words of memory that are stored during the computation of the (approximate) solution in the worst case for any data stream. We count the number of words instead of the number of bits in order to simplify the notation as we assume that all quantities, including elements, values, and also auxiliary data such as hash values etc., can be stored in one word of memory.

However, once we start to throw away information about the stream, an algorithm can no longer determine the exact weight of each element. In particular, deciding whether a weight reaches the threshold T becomes hard. This intuitive argument can be formalized using a simple reduction from the *set disjointness problem* [19, 22, 25], which yields that any *randomized* algorithm that computes Π with reasonable probability must store at least $\Omega(m)$ bits for any threshold T. This fundamental limitation forces us to resort to randomized approximation algorithms. Hence, instead of trying to compute the correct solution Π, we focus on algorithms that approximate Π in the sense that the computed weights may deviate from the true weights, but the deviation is bounded with a tunable probability. Consequently, the elements in the output set may not be identical to those in Π. The relaxed version of the heavy distinct hitter problem has two additional parameters and is defined as follows.

Definition 2.1. ***Approximate Heavy Distinct Hitter Problem.*** *Given a stream $\mathcal{S}$, parameters $\varepsilon, \delta \in (0, 1)$, and a threshold value $T > 0$, output a set $\mathcal{L}$ of pairs $(e, \tilde{w}(e))$ for which it holds that*

(1) if element e is in $\mathcal{L}$, then $w(e) \geq (1 - \varepsilon)T$

(2) if element e is not in $\mathcal{L}$, then $w(e) < (1 + \varepsilon)T$

(3) for all $(e, \tilde{w}(e)) \in \mathcal{L}$ we have that $|w(e) - \tilde{w}(e)| \leq \varepsilon T$

with probability at least $1 - \delta$.

The parameter ε specifies that any element e for which $w(e) \in [(1 - \varepsilon)T, (1 + \varepsilon)T)$ may or may not be added to $\mathcal{L}$, whereas the decision must be correct for all other elements (Condition (1) and Condition (2)). Condition (3) upper bounds the error in the estimated weights, i.e., it imposes a constraint on the accuracy. It is convenient to take εT as the upper bound on the error for the following reason. If the error in each estimated weight $\tilde{w}(e)$ is at most εT (with a certain probability), then we can simply add exactly those elements e to $\mathcal{L}$ for which $\tilde{w}(e) \geq T$, and both Condition (1) and Condition (2) hold as well. The second parameter δ is the error probability of the algorithm, i.e., we consider *Monte Carlo* algorithms. We say that an algorithm *(ε, δ)-approximates* the correct solution Π if its output $\mathcal{L}$ meets all the requirements of Definition 2.1.

It may suffice to guarantee a bound on the error that is proportional to the true weight. In this case, we substitute the *strong accuracy* condition (Condition (3)) with the following *weak accuracy* condition:

(3') for all $(e, \tilde{w}(e)) \in \mathcal{L}$ we have that $|w(e) - \tilde{w}(e)| \leq \varepsilon w(e)$.

[2]In contrast, an element e is considered a *heavy hitter* (see, e.g., [9]) if the sum of values that occur together with e as a pair in the stream is large.

As we will see, there is a fundamental difference between strong and weak accuracy in that weak accuracy can be achieved using less space. The disadvantage of the weak accuracy condition is that it gives weaker guarantees on the correctness of an output list sorted according to the estimated weights, i.e., it is more likely that $w(e) > w(e')$ but $\tilde{w}(e) < \tilde{w}(e')$ for some elements $e, e' \in E_{\mathcal{S}}$. Note that strong accuracy implies that Condition (3') holds for $\varepsilon' := \frac{\varepsilon}{1-\varepsilon}$: Since all elements in $\mathcal{L}$ have a weight of at least $(1-\varepsilon)T$ and the error in the estimated weights is upper bounded by εT, we immediately get that $|w(e) - \tilde{w}(e)| \leq \frac{\varepsilon}{1-\varepsilon} w(e) = \varepsilon' w(e)$ for all $e \in \mathcal{L}$. Therefore, we focus on strong accuracy and discuss potential space optimizations for weak accuracy along the way.

In the traditional heavy hitter problem, the threshold T is simply a fraction of the length n of the data stream. This definition has also been used in the context of the heavy distinct hitter problem [26]. However, since we are interested in finding the elements that contribute the most to the "total weight" m, it is more natural to define T as a fraction of m. Another motivation for this definition is that in the case of applications where n constitutes the size of the considered window, no element may have a weight that is a significant fraction of n, yet some elements may have a large weight with respect to m if the number of duplicates in the data stream is large. Hence, we strive to identify the elements whose weight is at least $T = \phi m$ for a parameter $\phi \in (0, 1)$.[3]

3. ALGORITHMS

An essential ingredient for both algorithms discussed in this section are pseudo-random hash functions mapping elements or element-value pairs to a particular image uniformly at random. We assume that all hash functions are independent. In the first part of this section, a simple sampling-based algorithm is presented, followed by a discussion of a sketch-based algorithm in Section 3.2.

3.1 Sampling-Based Heavy Distinct Hitter Algorithm

A straightforward approach to reducing the space complexity is to randomly drop element-value pairs in the stream, i.e., merely a small, random sample of the entire stream is stored. The basic idea is that any element e that ends up in the sample together with many different values probably occurs in the stream with a large number of distinct values if each element-value pair has a small probability of being sampled. While this simple trick suffices to detect heavy distinct hitters with reasonable probability, the errors in the estimated weights may still be large. This problem is addressed by generating several samples in parallel and extracting a more accurate estimator from them.

3.1.1 Description

The most crucial parameter of the presented sampling algorithm, referred to as $\mathcal{A}^{sample}$, is the *sampling probability* $p \in (0, 1)$ as it must be large enough to ensure that all heavy distinct hitters are sampled frequently, but small enough to make sure that the space complexity remains low. As mentioned above, several samples are required in order to bound the error in the estimated weights. Thus, the second parameter of $\mathcal{A}^{sample}$ is the number r of independent samples used by the algorithm.

[3]It is worth noting that, since $m \leq n$, a solution for $T = \phi m$ also contains all heavy distinct hitters for $T = \phi n$.

Algorithm 1 $\mathcal{A}^{sample}$: Process element-value pair (e, v).

```
for $i := 1, \dots, r$ do
    if $h_i((e, v)) < p$ and $(e, v) \notin \mathcal{R}_i$ then
        $\mathcal{R}_i := \mathcal{R}_i \cup \{(e, v)\}$;
        $\tilde{w}_i(e) := |\{(e', v') \in \mathcal{R}_i \mid e = e'\}|/p$;
    end if
end for
$\tilde{w}(e) := \text{median}(\tilde{w}_1(e), \dots, \tilde{w}_r(e))$;
if $\tilde{w}(e) \geq T$ then
    $\mathcal{L} := (\mathcal{L} \setminus \{(e, \cdot)\}) \cup \{(e, \tilde{w}(e))\}$;
end if
```

For all $i \in \{1, \dots, r\}$, a pseudo-random hash function $h_i : (E \times V) \to [0, 1]$, mapping element-value pairs to a value in the range $[0, 1]$ uniformly at random, is associated with the i^{th} sample $\mathcal{R}_i$. These hash functions are used to determine whether an element-value pair is added to the sample as follows. When processing an element-value pair (e, v) in the stream, it is added independently to each sample $\mathcal{R}_i$ if $h_i((e, v)) < p$ and it has not been added before.[4] An estimate $\tilde{w}_i(e)$ of element e's weight is computed for each $i \in \{1, \dots, r\}$, which is simply the number of distinct values that occur together with e in the sample divided by the sampling probability p. The final estimate $\tilde{w}(e)$ is the median of these r estimates. An element, together with its estimated weight, is added to the output set $\mathcal{L}$ if the estimated weight is at least $T = \phi m$. The issue that m, and hence T, is not known a priori will be discussed later. The algorithm is summarized in Algorithm 1.

3.1.2 Analysis

In order to simplify the analysis, we assume in the following that $\varepsilon \leq 1/2$. Moreover, the parameters of $\mathcal{A}^{sample}$ are set to $p := \frac{4e}{(\varepsilon\phi)^2 m}$ and $r := 2\lceil \log(\frac{4}{\phi\delta}) \rceil - 1$. As a first step, Lemma 3.1 shows that each estimate $\tilde{w}_i(e)$ is statistically unbiased, i.e., each estimate is $w(e)$ in expectation. In addition, the lemma upper bounds the variance of $\tilde{w}_i(e)$.

LEMMA 3.1. *For all $i \in \{1, \dots, r\}$ and $e \in E_{\mathcal{S}}$, it holds that $\mathbb{E}[\tilde{w}_i(e)] = w(e)$ and $Var(\tilde{w}_i(e)) < w(e)\frac{(\varepsilon\phi)^2 m}{4e}$.*

PROOF. Consider any sampled element e and an arbitrary $i \in \{1, \dots, r\}$. Let D_i denote the number of distinct values stored in $\mathcal{R}_i$ that form an element-value pair with e. We have that $\mathbb{E}[D_i] = p \cdot w(e)$ and thus $\mathbb{E}[\tilde{w}_i(e)] = \mathbb{E}[D_i/p] = w(e)$. The variance is $Var(\tilde{w}_i(e)) = Var(D_i/p) = Var(D_i)/p^2 = w(e)(1-p)/p < w(e)\frac{(\varepsilon\phi)^2 m}{4e}$. □

While the estimated weights of the heavy distinct hitters must be fairly accurate, the error for the remaining elements merely has to be small enough to ensure that they are not mistakenly considered heavy distinct hitters. In order to formalize this rule, we partition the set $E_{\mathcal{S}}$ of elements into disjoint classes $\mathcal{C}_j$, $j \in \{0, \dots, \lceil \log(\phi m) \rceil - 1\}$, as follows. An element belongs to class $\mathcal{C}_0$ if $w(e) \geq \frac{\phi m}{2}$. For each element

[4]Note that the pseudo-randomness of the hash functions implies that an element-value pair (e, v) that is not added to some sample $\mathcal{R}_i$ will not be added to $\mathcal{R}_i$ at any later point in time.

$e \in \mathcal{C}_0$ an upper bound of $b_0 := \varepsilon\phi m$ is imposed on the error of its estimated weight. We define that $e \in \mathcal{C}_j$ for any $j \geq 1$ if $w(e) \in \left[\frac{\phi m}{2^{j+1}}, \frac{\phi m}{2^j}\right)$. For each element in any such class, we require that the error in the estimated weight is bounded by $b_1 := b_2 := \ldots := \frac{\phi m}{2}$. The following lemma reveals that the probability that $\tilde{w}(e)$ deviates from $w(e)$ by b_j or more is small for any element $e \in C_j$, and also that this probability becomes exponentially smaller as j increases.

Lemma 3.2. *For each element e in any class $\mathcal{C}_j$ it holds that $\mathbb{P}[|\tilde{w}(e) - w(e)| \geq b_j] < \left(\frac{\phi\delta}{4}\right)^{j+1}$.*

Proof. First, consider the case $j = 0$. According to Lemma 3.1, the variance of each estimate $\tilde{w}_i(e)$ is bounded by $Var(\tilde{w}_i(e)) < w(e)\frac{(\varepsilon\phi)^2 m}{4e} \leq \frac{(\varepsilon\phi m)^2}{4e}$ for all $i \in \{1, \ldots, r\}$. By applying Chebychev's inequality, we immediately get that $\mathbb{P}[|\tilde{w}_i(e) - w(e)| \geq \varepsilon\phi m] < \frac{1}{4e} = \frac{1}{2^{j+2}e}$. For $j > 0$, we have that $Var(\tilde{w}_i(e)) \leq w(e)\frac{(\varepsilon\phi)^2 m}{4e} < \frac{(\varepsilon\phi m)^2}{2^{j+2}e}$ and thus $\mathbb{P}[|\tilde{w}_i(e) - w(e)| \geq (\phi m)/2] < \frac{4\varepsilon^2}{2^{j+2}e} \leq \frac{1}{2^{j+2}e}$, using the assumption that $\varepsilon \leq 1/2$. Hence, we can conclude that

$$p_j := \mathbb{P}[|\tilde{w}_i(e) - w(e)| \geq b_j] < \frac{1}{2^{j+2}e} \qquad (1)$$

for any element $e \in C_j$, $j \in \{0, \ldots, \lceil\log(\phi m)\rceil - 1\}$.

Since the final estimate $\tilde{w}(e)$ is the median of r estimates, it is only possible that $|\tilde{w}(e) - w(e)| \geq b_j$ if more than $(r-1)/2$ estimates deviate from $w(e)$ by at least b_j. This probability is upper bounded by

$$\begin{aligned}
\mathbb{P}[|\tilde{w}(e) - w(e)| \geq b_j] &\leq \sum_{k=(r+1)/2}^{r} \binom{r}{k} p_j^k (1-p_j)^{r-k} \\
&\leq \binom{r}{(r+1)/2} p_j^{(r+1)/2} \\
&< (2e)^{(r+1)/2} p_j^{(r+1)/2} \\
&\overset{(1)}{<} \left(\frac{1}{2^{j+1}}\right)^{\log(4/(\phi\delta))} = \left(\frac{\phi\delta}{4}\right)^{j+1}.
\end{aligned}$$

□

Given this lemma, we are in the position to prove the main result in this section, which states that $\mathcal{A}^{sample}$ indeed computes an (ε, δ)-approximation. In addition, Theorem 3.3 gives a bound on the space complexity of $\mathcal{A}^{sample}$ that holds *with high probability.*[5]

Theorem 3.3. *If $p := \frac{4e}{(\varepsilon\phi)^2 m}$ and $r := 2\lceil\log(\frac{4}{\phi\delta})\rceil - 1$, $\mathcal{A}^{sample}$ (ε, δ)-approximates the correct solution $\Pi = \{(e, w(e)) \mid w(e) \geq T\}$. The space complexity is*

$$\mathcal{O}\left(\left(1 + \varepsilon\phi\sqrt{\log m}\right)\frac{\log(\frac{1}{\phi\delta})}{(\varepsilon\phi)^2}\right)$$

with high probability.

Proof. As there are at most $\frac{2^{j+1}}{\phi}$ elements in class $\mathcal{C}_j$, it follows from Lemma 3.2 that the probability that *any* element in this class does not satisfy the required bound is $\frac{2^{j+1}}{\phi}\left(\frac{\phi\delta}{4}\right)^{j+1} \leq \left(\frac{1}{2}\right)^{j+1}\delta$. Hence, by a union bound, the probability that *any* element in *any* class does not satisfy the required bound is upper bounded by δ. Since the error in $\tilde{w}(e)$ for all $e \in \mathcal{C}_0$ is upper bounded by $\varepsilon\phi m = \varepsilon T$ and the estimated weight is smaller than T for all other elements, $\mathcal{A}^{sample}$ (ε, δ)-approximates Π as claimed.

At most $\frac{1}{(1-\varepsilon)\phi}$ element-weight pairs are added to $\mathcal{L}$, i.e., the space complexity for storing $\mathcal{L}$ is $\mathcal{O}(1/\phi)$ provided that ε is bounded away from 1. Let R denote the total number of sampled element-value pairs.[6] The simple bound $2\lceil\log(\frac{4}{\phi\delta})\rceil - 1 \geq 3$ implies that $\mathbb{E}[R] = rpm \geq \frac{12e}{(\varepsilon\phi)^2}$. Using a Chernoff bound, we get that

$$\begin{aligned}
\mathbb{P}\left[R > \left(1 + \frac{\varepsilon\phi}{\sqrt{3e}}\sqrt{\lambda \ln m}\right)\mathbb{E}[R]\right] &\leq e^{-\frac{(\varepsilon\phi)^2}{12e}(\lambda \ln m)\mathbb{E}[R]} \\
&\leq e^{-\lambda \ln m} = \frac{1}{m^\lambda}.
\end{aligned}$$

Since $\mathbb{E}[R] = rpm < \frac{8e}{(\varepsilon\phi)^2}\lceil\log(\frac{4}{\phi\delta})\rceil$, the claimed bound on the space complexity follows. □

A nice property of $\mathcal{A}^{sample}$ is that it can be adapted to guarantee only weak accuracy at a lower space complexity. Not suprisingly, this reduction of the space complexity is achieved by reducing the sampling probability p, which yields the following result.

Theorem 3.4. *If $p := \frac{4e}{(1-\varepsilon)\varepsilon^2\phi m}$ and $r := 2\lceil\log(\frac{4}{\phi\delta})\rceil - 1$, $\mathcal{A}^{sample}$ (ε, δ)-approximates the correct solution $\Pi = \{(e, w(e)) \mid w(e) \geq T\}$ guaranteeing weak accuracy. The space complexity is*

$$\mathcal{O}\left(\left(1 + \varepsilon\sqrt{(1-\varepsilon)\phi\log m}\right)\frac{\log(\frac{1}{\phi\delta})}{(1-\varepsilon)\varepsilon^2\phi}\right)$$

with high probability.

Proof. Due to the smaller sampling probability p, the variance increases slightly, i.e., we have that $Var(\tilde{w}_i(e)) < w(e)\frac{(1-\varepsilon)\phi\varepsilon^2 m}{4e}$ for all $i \in \{1, \ldots, r\}$ and $e \in E_{\mathcal{S}}$.

We will now show that Lemma 3.2 still holds if we split the class $\mathcal{C}_0$ into two disjoint classes $\mathcal{C}_0^*$ and $\mathcal{C}_0$ and define $b_0^*(e) := \varepsilon w(e)$ for all $e \in \mathcal{C}_0^*$. An element belongs to class $\mathcal{C}_0^*$ if its weight is at least $(1-\varepsilon)\phi m$, i.e., exactly those elements are in $\mathcal{C}_0^*$ that are allowed to occur in $\mathcal{L}$. Each other element e, whose weight is in the range $[\frac{\phi m}{2}, (1-\varepsilon)\phi m)$, remains in class $\mathcal{C}_0$ for which $b_0 := \varepsilon\phi m$.

Consider any element $e \in \mathcal{C}_0^*$. It holds that $Var(\tilde{w}_i(e)) < w(e)\frac{(1-\varepsilon)\phi\varepsilon^2 m}{4e} \leq \frac{(\varepsilon w(e))^2}{4e}$. Thus, the probability that $\tilde{w}_i(e)$ deviates from $w(e)$ by at least $b_0^*(e) = \varepsilon w(e)$ is lower than $1/(4e)$. For each element $e \in \mathcal{C}_0$ we have that $Var(\tilde{w}_i(e)) < w(e)\frac{(1-\varepsilon)\phi\varepsilon^2 m}{4e} < \frac{(\varepsilon\phi m)^2}{4e}$ and thus $\mathbb{P}[|\tilde{w}_i(e) - w(e)| \geq b_0] < 1/(4e)$. Hence, for each element in either $\mathcal{C}_0^*$ or $\mathcal{C}_0$ the failure probability is lower than $1/(4e) = 1/(2^{j+2}e)$. For all $e \in \mathcal{C}_j$, $j > 0$, and $i \in \{1, \ldots, r\}$, it holds that $Var(\tilde{w}_i(e)) < w(e)\frac{(1-\varepsilon)\phi\varepsilon^2 m}{4e} < \frac{(\varepsilon\phi m)^2}{2^{j+2}e}$, and thus $\mathbb{P}[|\tilde{w}_i(e) - w(e)| \geq b_j = \phi m/2] < 1/(2^{j+2}e)$, as $\varepsilon \leq 1/2$ by assumption. Using the same techniques as in Lemma 3.2, we again get that $\mathbb{P}[|\tilde{w}(e) - w(e)| \geq b_j] < \left(\frac{\phi\delta}{4}\right)^{j+1}$ for all j.

The proof of correctness is now identical to the proof of Theorem 3.3, and the space complexity is derived analogously. □

[5]For an input parameter m, an event holds *with high probability* if the probability that it occurs is at least $1 - 1/m^\lambda$, where $\lambda \geq 1$ is a parameter of the algorithm or the analysis.

[6]Note that the same stream element-value pair may occur in more than one sample $\mathcal{R}_i$. Given that p is small, this does not increase the space complexity substantially.

We see that the space complexity can be improved roughly by a factor of $(1-\varepsilon)/\phi$ if weak accuracy suffices.

3.1.3 Discussion

It is easy to show that the bounds on the space complexity in Theorem 3.3 and Theorem 3.4 are asymptotically optimal up to logarithmic factors.

THEOREM 3.5. *Any algorithm $\mathcal{A}$ that (ε,δ)-approximates $\Pi = \{(e, w(e)) \mid w(e) \geq T\}$ guaranteeing weak accuracy must store $\Omega\big(\frac{1}{\varepsilon^2\phi}\big)$ bits. If strong accuracy is required, any algorithm $\mathcal{A}$ that (ε,δ)-approximates Π must store $\Omega\big(\frac{1}{(\varepsilon\phi)^2}\big)$ bits.*

PROOF. Both bounds follow from simple reduction arguments. First, we consider the space required to guarantee weak accuracy. Assume that the heavy distinct hitters are *given*, i.e., it remains for algorithm $\mathcal{A}$ to estimate their weights. Without loss of generality, it is further assumed that the approximation of the weight of each heavy distinct hitter is independent. As there may be $\Omega(1/\phi)$ heavy distinct hitters, algorithm $\mathcal{A}$ must compute the number of distinct values for $\Omega(1/\phi)$ elements independently, implying that algorithm $\mathcal{A}$ must store $\Omega(1/\phi)$ times as many bits as are required to compute an accurate estimate of the weight of a single heavy distinct hitter. Since any algorithm computing an (ε,δ)-approximation of the number of distinct elements must store $\Omega(1/\varepsilon^2)$ bits if m is sufficiently large [17], $\Omega(1/\varepsilon^2)$ bits are also needed to get an accurate estimate of the weight of each heavy distinct hitter, which proves the claimed bound.

An algorithm $\mathcal{A}$ that (ε,δ)-approximates Π guaranteeing strong accuracy can be used to (ε',δ)-approximate m, where $\varepsilon' := \varepsilon\phi$, as follows. Each item x in the stream is converted into the element-value pair (e,x) and then processed by algorithm $\mathcal{A}$. In the end, $\mathcal{L}$ will contain $(e,\tilde{w}(e))$ for which it holds that $|\tilde{w}(e) - w(e)| = |\tilde{w}(e) - m| \leq \varepsilon(\phi m) = \varepsilon' m$ with probability at least $1-\delta$. Again using the result that $\Omega(1/\varepsilon'^2)$ bits are required to (ε',δ)-approximate the number of distinct elements, we conclude that algorithm $\mathcal{A}$ must store at least $\Omega(1/(\varepsilon\phi)^2)$ bits. $\square$

A minor shortcoming of $\mathcal{A}^{sample}$ is that the bound on the space complexity is probabilistic. This issue can be overcome by fixing the maximum number of sampled elements, e.g., by setting it to a small multiple of the expected number, which slightly increases the failure probability of the algorithm. A more critical problem is that m is unknown. As mentioned earlier, m can be approximated fairly efficiently. Given an estimate $\tilde{m}$ that lies in the range $[(1-\rho)m, (1+\rho)m]$ with reasonable probability, p is computed using the estimate $\tilde{m}/(1+\rho)$. This estimate is appropriate because the claimed error bounds on the weights with respect to the true m may be violated if m is overestimated. Thus, p may be too large by a factor of $(1+\rho)/(1-\rho)$, and the space complexity increases by the same factor. Naturally, the estimate $\tilde{m}$ and consequently the sampling probability p changes as elements are processed, i.e., it is necessary to iterate over the sampled elements and drop all the elements from each sample $\mathcal{R}_i$ if $h_i(e) < p$ does not hold anymore [23], in particular if intermediate results are required. In order to minimize the number of iterations, the estimate for m may only be increased if, e.g., $\tilde{m}$ reaches the next power of 2, which also leads to an increase in the space complexity.

Apparently, all these actions have a negative impact on the space complexity or the failure probability of the entire procedure. An obvious question is whether there are other approaches that are not affected by these problems. This issue is addressed in the following section, where an algorithm is presented whose parameters do no not depend linearly on m, which appears to be inherent to sampling-based approaches.

3.2 Sketch-Based Heavy Distinct Hitter Algorithm

A common technique for solving stream processing problems is to compute a so-called *sketch*, or *synopsis*, which is a space-efficient summary of a stream. The main difference to sampling is that typically the entire stream is added to the sketch (i.e., no input is simply dropped). Finding a suitable sketch for the heavy distinct hitter problem is challenging because only *distinct* values must be counted, which means that a heavy hitter approach cannot be used. The sketch introduced in this section solves this problem by employing pairs of distinct counting primitives capable of computing an estimate of the number of distinct items inserted. An element-value pair is processed by randomly inserting it into one of each pair of distinct counting primitives. The intuition is that updating the same counting primitive of each pair when processing $(e,\cdot)$ produces an imbalance between the estimates of the primitives, which can be exploited to estimate $w(e)$.

3.2.1 Description

The basic building block of our sketch is a distinct counting primitive C, which offers two functions: (1) *insert*(x) processes the data item x and (2) *getNumberDistinct*$()$ returns an estimate of the (current) number of distinct inserted items. Any distinct counting primitive that satisfies the following criteria can be used: It can be stored using a constant number of words, and if m distinct items are inserted, *getNumberDistinct*$()$ returns the correct number m in expectation and the variance is αm^2 for some constant $\alpha > 0$. It can be shown that this bound on the variance is asymptotically optimal if the space complexity of C is constant [17]. Furthermore, we require that the estimate does not change when some item is inserted repeatedly. Several distinct counting algorithms described in the literature meet these requirements. We will discuss such algorithms in more detail later.

The algorithm, called $\mathcal{A}^{sketch}$, is similar to $\mathcal{A}^{sample}$ in that it also computes r estimates and returns the median as the final estimate. The second parameter s of $\mathcal{A}^{sketch}$ determines how many pairs of distinct counting primitives are used to compute each estimate. For a certain $i \in \{1,\ldots,r\}$ and $j \in \{1,\ldots,s\}$, the two distinct counting primitives that form the j^{th} pair used for the i^{th} estimate are denoted by C^0_{ij} and C^1_{ij}. Each stream item (e,v) is processed by inserting it into one distinct counting primitive of each pair. A pseudorandom hash function $h_{ij} : E \to \{0,1\}$, hashing each element e to 0 or 1 with equal probability, determines whether (e,v) is inserted into C^0_{ij} or C^1_{ij} irrespective of $v \in V$. After this insertion process, the updated estimate $\tilde{w}(e)$ can be computed and, as in $\mathcal{A}^{sample}$, $(e,\tilde{w}(e))$ is added to the output set $\mathcal{L}$ if $\tilde{w}(e) \geq T$ (i.e., T must again be approximated). The steps of $\mathcal{A}^{sketch}$ are given in Algorithm 2.

It remains to specify how $\tilde{w}(e)$ is determined. As men-

Algorithm 2 $\mathcal{A}^{sketch}$: Process element-value pair (e, v).

```
for $i = 1, \ldots, r$ do
  for $j = 1, \ldots, s$ do
    $C_{ij}^{h_{ij}(e)}$.insert($(e, v)$);
  end for
end for
$\mathcal{L} := \mathcal{L} \setminus \{(e, \cdot)\}$;
$\tilde{w}(e) :=$ getEstimate($e$);
if $\tilde{w}(e) \geq T$ then
  $\mathcal{L} := \mathcal{L} \cup \{(e, \tilde{w}(e))\}$;
end if
```

tioned before, it is the median of estimates $\tilde{w}_1(e), \ldots, \tilde{w}_r(e)$. Each estimate $\tilde{w}_i(e)$, $i \in \{1, \ldots, r\}$, is computed as follows. For all s pairs of distinct counting primitives, the difference between the estimated number of distinct insertions into $C_{ij}^{h_{ij}(e)}$ and $C_{ij}^{1-h_{ij}(e)}$ is computed, and $\tilde{w}_i(e)$ is simply set to the average of these differences. This procedure, called *getEstimate*(e), is summarized in Algorithm 3.

3.2.2 Analysis

The analysis of algorithm $\mathcal{A}^{sketch}$ basically follows the same lines as the analysis of $\mathcal{A}^{sample}$. For the sake of simplicity, we assume in this section that $\varepsilon \leq 1/2 - c$ for some constant $c > 0$ (this assumption is used in the proof of Theorem 3.7). In the following, we will slightly abuse our notation and consider $C_{ij}^{h_{ij}(e)}$ a random variable whose value is the corresponding distinct counting primitive's estimate of the number of distinct inserted items. The key result, which is proved in the following lemma, is that $C_{ij}^{h_{ij}(e)} - C_{ij}^{1-h_{ij}(e)}$ is an unbiased estimator of the weight $w(e)$ of element e, and its variance is in the order of m^2.

LEMMA 3.6. *For all $i \in \{1, \ldots, r\}$, $j \in \{1, \ldots, s\}$, and $e \in E_{\mathcal{S}}$, it holds that $\mathbb{E}[C_{ij}^{h_{ij}(e)} - C_{ij}^{1-h_{ij}(e)}] = w(e)$ and $Var(C_{ij}^{h_{ij}(e)} - C_{ij}^{1-h_{ij}(e)}) \leq (1+\alpha)m^2 - mw(e)$.*

PROOF. For all $i \in \{1, \ldots, r\}$, $j \in \{1, \ldots, s\}$, and $q \in \{0, 1\}$, let the random variable $C_{ij}^q(t)$ denote the return value of C_{ij}^q.*getNumberDistinct*() after t distinct items have been inserted. Recall that $\mathbb{E}[C_{ij}^q(t)] = t$ and $Var(C_{ij}^q(t)) = \alpha t^2$ according to our requirements of distinct counting primitives given in Section 3.2. For any two elements e and e', we define

$$s(e, e') := \begin{cases} 1 & \text{if } h_{ij}(e) = h_{ij}(e'), \\ 0 & \text{else.} \end{cases}$$

It holds that

$$\begin{aligned} \mathbb{E}\left[C_{ij}^{h_{ij}(e)}\right] &= \mathbb{E}\Big[C_{ij}^{h_{ij}(e)}\Big(w(e) + \sum_{e' \in E_{\mathcal{S}} \setminus \{e\}} s(e, e')w(e')\Big)\Big] \\ &= w(e) + \frac{m - w(e)}{2} = \frac{m + w(e)}{2}. \end{aligned}$$

Similarly, we get that $\mathbb{E}[C_{ij}^{1-h_{ij}(e)}] = \frac{m-w(e)}{2}$, and thus $\mathbb{E}[C_{ij}^{h_{ij}(e)} - C_{ij}^{1-h_{ij}(e)}] = w(e)$ as claimed.

$\mathbb{E}[C_{ij}^q(t)] = t$ and $Var(C_{ij}^q(t)) = \alpha t^2$ together imply that $\mathbb{E}[(C_{ij}^q(t))^2] = (1+\alpha)t^2$. Let p_t denote the probability that the total number of distinct elements hashed to $C_{ij}^{h_{ij}(e)}$ is t.

Algorithm 3 getEstimate(e): Compute the estimated weight of element e.

```
for $i = 1, \ldots, r$ do
  $\tilde{w}_i := 0$;
  for $j = 1, \ldots, s$ do
    $\tilde{w}_i := \tilde{w}_i + C_{ij}^{h_{ij}(e)}$.getNumberDistinct()
        $- C_{ij}^{1-h_{ij}(e)}$.getNumberDistinct();
  end for
  $\tilde{w}_i := \tilde{w}_i / s$;
end for
return median($\{\tilde{w}_1, \ldots, \tilde{w}_r\}$);
```

We have that

$$\begin{aligned} \mathbb{E}\left[\left(C_{ij}^{h_{ij}(e)}\right)^2\right] &= \sum_{t=w(e)}^{m} \mathbb{E}\left[\left(C_{ij}^{h_{ij}(e)}(t)\right)^2\right] p_t \\ &= (1+\alpha) \sum_{t=w(e)}^{m} t^2 p_t \\ &\leq \frac{1+\alpha}{2}\left(m^2 + w(e)^2\right). \end{aligned} \quad (2)$$

The last inequality can be explained as follows. Due to the quadradic dependency on t, the sum is maximized if the probability that $t = m$ is maximized, i.e., the number of different elements is minimized. Assume that there is just one other element e' whose weight is $w(e') = m - w(e)$. In this case, $p_{w(e)} = p_m = 1/2$ and we get exactly $\mathbb{E}[(C_{ij}^{h_{ij}(e)})^2] = \frac{1+\alpha}{2}(m^2 + w(e)^2)$. Note that this bound also holds if e is the only element, which implies that $\mathbb{E}[(C_{ij}^{h_{ij}(e)})^2] = (1+\alpha)m^2$. If there are more elements, p_m is reduced, and $\mathbb{E}[(C_{ij}^{h_{ij}(e)})^2]$ becomes smaller as a result. The same argument also applies to $C_{ij}^{1-h_{ij}(e)}$ for which it holds that

$$\mathbb{E}\left[\left(C_{ij}^{1-h_{ij}(e)}\right)^2\right] \leq \frac{1+\alpha}{2}(m - w(e))^2. \quad (3)$$

In order to bound the variance, we need a lower bound on the covariance:

$$\begin{aligned} Cov\left(C_{ij}^{h_{ij}(e)}, C_{ij}^{1-h_{ij}(e)}\right) &\geq -\mathbb{E}\left[C_{ij}^{h_{ij}(e)}\right]\mathbb{E}\left[C_{ij}^{1-h_{ij}(e)}\right] \\ &= -\frac{m^2 - w(e)^2}{4}. \end{aligned} \quad (4)$$

Given these bounds, we get the claimed bound on the variance of $\Delta C_{ij}(e) := C_{ij}^{h_{ij}(e)} - C_{ij}^{1-h_{ij}(e)}$ as follows.

$$\begin{aligned} Var(\Delta C_{ij}(e)) &= Var\left(C_{ij}^{h_{ij}(e)}\right) + Var\left(C_{ij}^{1-h_{ij}(e)}\right) \\ &\quad -2Cov\left(C_{ij}^{h_{ij}(e)}, C_{ij}^{1-h_{ij}(e)}\right) \\ &\overset{(2,3,4)}{\leq} \frac{1+\alpha}{2}\left(m^2 + w(e)^2\right) - \left(\frac{m + w(e)}{2}\right)^2 \\ &\quad + \frac{1+\alpha}{2}(m - w(e))^2 - \left(\frac{m - w(e)}{2}\right)^2 \\ &\quad + \frac{m^2 - w(e)^2}{2} \\ &= (1+\alpha)m^2 + \alpha w(e)^2 - (1+\alpha)mw(e) \\ &\leq (1+\alpha)m^2 - mw(e). \end{aligned}$$

□

Considering that $\mathcal{O}(m)$ distinct element-value pairs are inserted into each distinct counting primitive, it is not surprising that the variance of the difference between two distinct counting primitives is in the order of m^2. The fact that the variance is not bounded by $\mathcal{O}(mw(e))$ (as in algorithm $\mathcal{A}^{sample}$) entails that the space complexity depends logarithmically on the number ℓ of distinct elements in the stream as Theorem 3.7 reveals.

THEOREM 3.7. *If* $r := 2\lceil \max\{\ln(\frac{4}{\phi\delta}), \frac{1}{2}\log_{1/(2\varepsilon)}(\frac{\phi\ell}{2})\}\rceil - 1$ *and* $s := \lceil \frac{(1+\alpha)2e^2}{(\phi\varepsilon)^2}\rceil$, $\mathcal{A}^{sketch}$ *(ε,δ)-approximates the correct solution* $\Pi = \{(e, w(e)) \mid w(e) \geq T\}$. *The space complexity is*

$$\mathcal{O}\left(\frac{\log(\frac{1}{\phi\delta}) + \log_{1/\varepsilon}(\phi\ell)}{(\phi\varepsilon)^2}\right).$$

PROOF. By assumption, repeated insertions of the same element-value pair do not have any effect on the distinct counting primitives. Therefore, we can assume without loss of generality that each element-value pair occurs only once in the stream.

Consider the time when a specific element e whose weight is at least $\frac{\phi m}{2}$ occurs the last time. At this point in time, e has been processed exactly $w(e)$ times. Each $\tilde{w}_i(e)$ is the average of s trials, which means that $\mathbb{E}[\tilde{w}_i(e)] = w(e)$ and $Var(\tilde{w}_i(e)) \leq \frac{(1+\alpha)m^2}{s} \leq \frac{(\varepsilon\phi m)^2}{2e^2}$ since at most m (distinct) element-value pairs have been processed. Consequently, using Chebychev's inequality, we get that $p' := \mathbb{P}[|\tilde{w}_i(e) - w(e)| > \varepsilon\phi m] \leq \frac{1}{2e^2}$. As in the analysis of algorithm $\mathcal{A}^{sample}$, the probability that the error of the final estimate $\tilde{w}(e)$ is at least $\varepsilon(\phi m)$ is upper bounded by the probability that more than $(r-1)/2$ estimates are off by at least $\varepsilon(\phi m)$, i.e.,

$$\begin{aligned}\mathbb{P}[|\tilde{w}(e) - w(e)| > \varepsilon\phi m] &\leq \binom{r}{(r+1)/2} p'^{(r+1)/2} \\ &\leq (2e)^{(r+1)/2} p'^{(r+1)/2} \leq \frac{\phi\delta}{4},\end{aligned}$$

where we used that $r \geq 2\lceil \ln(4/(\phi\delta))\rceil - 1$. By means of a union-bound argument, we see that the probability that $\tilde{w}(e)$ deviates form $w(e)$ by more than $\varepsilon(\phi m)$ for any element whose weight is at least $\frac{\phi m}{2}$ is upper bounded by $\delta/2$ because there are at most $2/\phi$ such elements. Thus, all elements whose weight is at least $\frac{\phi m}{2}$ satisfy the requirements with probability at least $1 - \delta/2$.

For the remaining elements it suffices to show that the error does not exceed $\frac{\phi m}{2}$, which ensures that none of these elements is erroneously considered a heavy distinct hitter. Since $p'' := \mathbb{P}[|\tilde{w}_i(e) - w(e)| > \phi m/2] \leq \frac{2\varepsilon^2}{e^2}$, the probability that $\tilde{w}(e)$ of any such element e is larger than $\frac{\phi m}{2}$ is

$$\begin{aligned}\mathbb{P}\left[\left|\tilde{w}(e) - w(e)\right| > \frac{\phi m}{2}\right] &\leq \binom{r}{(r+1)/2} p''^{(r+1)/2} \\ &\leq (2e)^{(r+1)/2} p''^{(r+1)/2} \\ &\leq \left(\frac{1}{e}\right)^{(r+1)/2} (4\varepsilon^2)^{(r+1)/2} \\ &\leq \frac{\phi\delta}{4}(2\varepsilon)^{\log_{1/(2\varepsilon)}(\phi\ell/2)} \qquad (5) \\ &= \frac{\delta}{2\ell}.\end{aligned}$$

In Inequality (5) we used that $r \geq 2\ln(4/(\phi\delta)) - 1$ and $r \geq \log_{1/(2\varepsilon)}(\phi\ell/2) - 1$. Again, using a union-bound argument, the probability that the estimated weight of any element exceeds $\frac{\phi m}{2}$ is upper bounded by $\delta/2$. Hence it follows that the estimates of all elements are as accurate as required with probability at least $1 - \delta$ as claimed.

The space complexity is $rs \cdot 2d + |\mathcal{L}|$ with d being the constant size of a distinct counting primitive. At most $\frac{1}{(1-\varepsilon)\phi}$ element-value pairs are added to $\mathcal{L}$, implying that $|\mathcal{L}| \in \mathcal{O}(1/\phi)$. Since $r := 2\lceil \max\{\ln(\frac{4}{\phi\delta}), \frac{1}{2}\log_{1/(2\varepsilon)}(\frac{\phi\ell}{2})\}\rceil - 1$, $s := \lceil \frac{(1+\alpha)2e^2}{(\phi\varepsilon)^2}\rceil$, and $\log_{1/(2\varepsilon)} x \leq \frac{1}{c}\log_{1/\varepsilon} x$ for any $x \geq 1$ due to the assumption that $\varepsilon \leq 1/2 - c$, the bound on the space complexity follows. $\square$

It is worth noting that (the less important case of) $\varepsilon \in [1/2, 1)$ can be handled by setting $r := 2\lceil \ln(\frac{\ell}{\delta})\rceil - 1$, which results in a space complexity of $\mathcal{O}\big(\frac{\log(\ell/\delta)}{(\phi\varepsilon)^2}\big)$.

3.2.3 Discussion

While the space complexity of $\mathcal{A}^{sample}$ is essentially constant, the space complexity of $\mathcal{A}^{sketch}$ depends logarithmically on ℓ. However, when taking a closer look at the parameter r in Theorem 3.7, it becomes apparent that $r = 2\lceil \ln(\frac{4}{\phi\delta})\rceil - 1$ unless ℓ is exceedingly large. For example, if $\phi = \delta = \varepsilon = 1/10$, r must be set to a larger value only if $\ell > 10^9$. Thus, the space complexity is constant for most practical purposes. What is more, if the space required for each distinct counting primitive is small—one possible implementation using little space is presented in Section 3.3—, the space complexities of the two algorithms are identical up to a small constant factor.

The advantage of $\mathcal{A}^{sketch}$ is that it does not rely on an accurate estimate of m. However, this increased robustness comes at a certain price: Due to the larger variance, algorithm $\mathcal{A}^{sketch}$ cannot be adapted for the weak accuracy constraint. Moreover, the computational cost is higher as many hash values have to be computed when processing an element-value pair. This processing cost can be reduced by caching the hash values of heavy or recently encountered elements, i.e., there is a trade-off between computational and space complexity.

3.3 Practical Evaluation

So far, we have discussed bounds on the space complexity and the accuracy of the proposed algorithms that hold regardless of the distribution of the input stream. Since worst-case distributions rarely occur in practice, it is worthwhile to investigate the performance of the algorithms when processing real network data. For this purpose, the algorithms have been implemented and tested using undirectional flow data captured at the edge between the IBM Research campus network and the Internet. The considered trace is a collection of more than 12 million flows recorded in five days in May 2009. Our focus is on finding the sources that connect to many distinct destination IP addresses. In total, there are more than 725,000 distinct source-destination pairs in the trace. Luckily, the trace is an ideal test candidate: Five machines scanned almost an entire 16-bit subnetwork during this time, which means that each of these machines is responsible for 8.8% of all distinct connections. The sum of distinct connections of the top 10 sources amounts to 64% of the total sum. The number of distinct connections of the other approximately $12,000$ sources follows a heavy-tail distribution.

Before discussing the main results, we briefly describe the distinct counting primitive used in the implementation of $\mathcal{A}^{sketch}$. Each inserted item is hashed uniformly at random to a value in the range $(0,1)$ and the k smallest hash values ever encountered are stored. If h_k is the k^{th} smallest hash value (i.e., the largest stored value), the estimated number of distinct items is $(k-1)/h_k$ [16]. When inserting m distinct items, it can be shown that $\mathbb{E}[(k-1)/h_k] = m$ and $Var((k-1)/h_k) = \frac{k-1}{k-2}(m^2-m) - m^2 < \frac{m^2}{k-2}$. Thus, a variance of (at most) αm^2 is achieved by storing $\frac{1}{\alpha}+2$ hash values. Given the factor $1+\alpha$ in the parameter s, $k=4$ minimizes the space requirements and is used in the implementation.[7]

After processing the data stream, both algorithms output the 10 elements with the largest estimated weights. We evaluate how many top 5 and top 10 sources are identified correctly, and also the error in the estimates. As the parameter r mainly helps to keep *all* errors bounded, it does not affect the *median error* significantly. For the sake of simplicity, we focus on this measure and state only average median errors over several runs using different hash functions and different $r \in \{1,3,5,7,9\}$. In order to compare the performance of $\mathcal{A}^{sample}$ and $\mathcal{A}^{sketch}$, they are both allowed to store a certain fraction of the total number of distinct connections. This constraint simplifies the sampling algorithm because maintaining an accurate estimate of m, as discussed in Section 3.1.3, is not required, which also means that $\mathcal{A}^{sample}$ is slightly favored. Since $\mathcal{A}^{sample}$ has the additional advantage that it must only store element-value pairs, instead of pairs of distinct counting primitives, we can expect $\mathcal{A}^{sample}$ to achieve more accurate results. By setting the memory budget to 10% of the total sum of distinct connections, $\mathcal{A}^{sketch}$ correctly finds all top 5 and top 10 sources, and the median error in the estimates of the top 5 and the top 10 sources is roughly 5% and 8%, respectively. The median error is still reasonably small when reducing the budget to 1% (19% for the top 5 and 27% for the top 10 sources), and all top 5 sources are identified correctly, but often only 6 or 7 of the correct top 10 sources are found. Given a memory budget of 1%, the median error of $\mathcal{A}^{sample}$ is around 5-6% for both the top 5 and top 10 sources, and all top 10 sources are in the output list, i.e., due to the lower variance, $\mathcal{A}^{sample}$ achieves a better accuracy particularly for the sources outside the top 5. Even a budget of 0.1% suffices for $\mathcal{A}^{sample}$ to identify at least 4 of the top 5 and 9 of the top 10 sources with a median error of about 20%. We conclude that both algorithms are capable of finding the heavy distinct hitters in the top 5 using little space; however, in this setting $\mathcal{A}^{sample}$ achieves a greater accuracy in the estimates, which is in accord with the theoretical analysis.

4. RELATED WORK

There is a large body of work on *streaming* (or *one-pass*) algorithms, which process data streams exactly once and in order. For a nice introduction to streaming algorithms, the interested reader is referred to [24]. As mentioned before, one of the most well-known and well-studied problems in the streaming model is computing the number m of distinct elements in a data stream. An elegant approach to approximate m is to map each element to a pseudo-random bit string and to store the largest index i where the first 1 occurs. Since we can expect that roughly $m/2^k$ elements have the first 1 at position k, i is approximately $\log m$ [13] (see also [4, 10, 12, 21] and references therein). The alternative technique outlined in the previous section is proposed and analyzed in [16]. It has been shown that any algorithm that outputs an estimate whose error is bounded by εm with reasonable probability must store $\Omega(\log m + \frac{1}{\varepsilon^2})$ bits [1, 17]. An algorithm that matches this bound has been proposed recently [21].

Alon et al. introduced the more general problem of approximating the *frequency moments* $F_k := \sum_{e_i \in E_S} m_i^k$, where m_i denotes the frequency of element e_i, of a data stream for any $k \geq 0$ [1]. Note that the number of distinct elements is the 0^{th} frequency moment. In a series of papers, tight bounds (up to polylogarithmic factors) on the space requirements have been proved for all k [1, 3, 5, 8, 18].

The elements that occur frequently in a data stream may also be of interest. It has been shown that elements whose frequencies exceed a certain threshold can be found efficiently [23] using the "sample and hold" technique [15]. The basic idea is to sample each element with a certain probability. Once an element is sampled, its frequency is maintained correctly from this point on by updating it whenever the same element occurs again in the stream. This technique can also be used to find flows in a network that use up at least a certain fraction of the bandwidth by randomly sampling bytes [11]. While "sample and hold" is an elegant and efficient technique to count frequencies, it is not as useful for detecting heavy distinct hitters. The problem is that the frequencies of sampled elements cannot be updated easily, i.e., a distinct counting primitive must be used for each element, and the sampling rate must be $\Omega\big(\frac{1}{m\varepsilon\phi}\big)$, otherwise too many distinct values may be missed before sampling an element for the first time. Since the error must be bounded by $\varepsilon(\phi m)$ and some elements may have a weight in the order of m, the space requirement for such a distinct counting primitive is $\Omega\big(\frac{1}{(\varepsilon\phi)^2}\big)$ bits. Therefore, the resulting (expected) space requirement is $\Omega\big(\frac{1}{(\varepsilon\phi)^3}\big)$ bits, which is worse than the bounds in Section 3.

The problem of finding heavy distinct hitters has also been studied in the literature, although the considered models and the problem definitions are not identical. The most relevant related work focuses on finding *superspreaders* [26], which are entities in a network that connect to many distinct destinations, i.e., superspreaders are heavy distinct hitters. The authors show that straightforward sampling can be used to identify superspreaders using little space. In contrast to this work, the threshold is defined as $T = \phi n$, i.e., the number of distinct destinations must be a fraction of the length of the entire data stream. Moreover, their goal is primarily to *detect* superspreaders without considering how many distinct destinations are contacted. In particular, for a parameter b, their sampling algorithm detects each superspreader with probability at least $1-\delta$, and a source that contacts at most T/b destinations is erroneously considered a superspreader with probability at most δ. By setting the sampling probability to an appropriate value, the space complexity of their algorithm is $\mathcal{O}\big(\frac{\log(1/\delta)}{\phi}\big(1+\frac{1}{(b-1)^2}\big)\big)$ in expectation. If we apply our model and require that Condition (1) and Condition (2) hold, and also that *all* superspreaders are found with probability $1-\delta$, the space complexity becomes

[7]Note that $k=3$ could have been used as well.

$\mathcal{O}\big(\frac{\log(1/(\phi\delta))}{\phi\varepsilon^2}\big)$. Note that algorithm $\mathcal{A}^{sample}$ achieves the same expected space complexity while additionally guaranteeing weak accuracy.

Numerous other techniques that are based on sampling have been proposed [7, 20, 27]. Cao et al. [7] focus on identifying heavy distinct hitters for a "moderately large" threshold T. They propose a two-phase filtering method using Bloom filters [6] whose purpose is to remove the majority of elements with small weights. The weight of the remaining elements is estimated using a thresholded bitmap. Since the computed weights are biased, the authors further introduce a simple technique for bias correction using unbiased weight estimates of a small random sample of elements. Zhao et al. [27] address the problem that some of the sampled element-value pairs may occur frequently, which entails that the data structure used to store the sampled pairs must process the same pairs again and again. As the arrival rate of element-value pairs may be significantly higher than the processing rate of this data structure, the sampling rate has to be small in order to ensure that the data structure is not overwhelmed. However, a small sampling rate results in low accuracy. The authors propose to use a Bloom filter to filter out element-value pairs that have been encountered before. Due to the possibility of hash collisions, the update procedure of the data structure must be modified to obtain unbiased weight estimates. Additionally, a more sophisticated approach is presented using a two-dimensional bitmap, which achieves more accurate results according to experiments using traces of real-world network traffic. A different approach to boost the performance of algorithms for the heavy distinct hitter problem is to use special associative memories [2].

In another work, the space complexity of finding the top-k heavy distinct hitters, i.e., the k elements with the largest weights, is studied. The authors consider SYN flooding attacks, where the weight of an element is defined as the number of half-open TCP connections [14]. Their model is somewhat more general in the sense that they can handle deletions, i.e., once a TCP connections is fully established, it no longer contributes to the weight of the (destination) element. The space complexity of their algorithm is $\mathcal{O}\big(\frac{m\log^2(n/\delta)\log^2 m}{w(e_k)\varepsilon^2}\big)$, where $w(e_k)$ denotes the k^{th} largest weight.[8] If $w(e_k) \in \Omega(\phi m)$, the space complexity becomes $\mathcal{O}\big(\frac{\log^2(n/\delta)\log^2 m}{\phi\varepsilon^2}\big)$, i.e., if n, m, and the weight of the top-k elements are large, the space complexities of the algorithms presented in Section 3 are significantly lower.

5. CONCLUSION

As we have seen, while there is no space-efficient solution that finds the *correct* set of heavy distinct hitters and the corresponding weights, there are approximation techniques that yield accurate results with high probability and that have a small memory footprint. In particular, we have studied two classic approaches in stream processing, sampling and computing a sketch of the data stream. Both techniques achieve a space complexity that is asymptotically optimal up to small logarithmic factors given a *strong accuracy* constraint. The proposed sampling-based algorithm is further (almost) optimal if a weaker accuracy constraint suffices, whereas the sketch-based algorithm has the advantage that it does not rely on an accurate estimate of the number of distinct items in the data stream. The practical study shows that the sampling-based algorithm slightly outperforms the sketch-based counterpart in that its returned estimates are more accurate given the same memory budget. However, both algorithms are able to detect the heavy distinct hitters even when given considerably less space than the theoretical upper bounds demand. Another strong point of the proposed algorithms is that they are intrinsically parallelizable as they compute sets of independent estimates. These results suggest that the proposed algorithms may indeed be valuable for various stream processing applications.

[8]The space complexity given in their paper depends on the size of the domains E and V. Their bound can be reduced to the stated bound by hashing each pair to a value in a domain of size polynomial in m.

6. REFERENCES

[1] N. Alon, Y. Matias, and M. Szegedy. The Space Complexity of Approximating the Frequency Moments. *Journal of Computer and System Sciences*, 58(1):137–147, 1999.

[2] N. Bandi, D. Agrawal, and A. El Abbadi. Fast Algorithms for Heavy Distinct Hitters using Associative Memories. In *Proc. 27th International Conference on Distributed Computing Systems (ICDCS)*, 2007.

[3] Z. Bar-Yossef, T. S. Jayram, R. Kumar, and D. Sivakumar. An Information Statistics Approach to Data Stream and Communication Complexity. *Journal of Computer and System Sciences*, 68(4):702–732, 2004.

[4] Z. Bar-Yossef, T. S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan. Counting Distinct Elements in a Data Stream. In *Proc. 6th International Workshop on Randomization and Approximation Techniques (RANDOM)*, pages 1–10, 2002.

[5] L. Bhuvanagiri, S. Ganguly, D. Kesh, and C. Saha. Simpler Algorithm for Estimating Frequency Moments of Data Streams. In *Proc. 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 708–713, 2006.

[6] B. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Communications of the ACM (CACM)*, 13:422–426, 1970.

[7] J. Cao, Y. Jin, A. Chen, T. Bu, and Z.-L. Zhang. Identifying High Cardinality Internet Hosts. In *Proc. 28th IEEE Conference on Computer Communications (INFOCOM)*, pages 810–818, 2009.

[8] A. Chakrabarti, S. Khot, and X. Sun. Near-Optimal Lower Bounds on the Multi-Party Communication Complexity of Set Disjointness. In *In Proc. 18th IEEE Conference on Computational Complexity (CCC)*, pages 107–117, 2003.

[9] M. Charikar, K. Chen, and M. Farach-Colton. Finding Frequent Items in Data Streams. *Theoretical Computer Science*, 312(1):3–15, 2004.

[10] M. Durand and P. Flajolet. LogLog Counting of Large Cardinalities. In *Proc. 11th Annual European Symposium on Algorithms (ESA)*, pages 605–617, 2003.

[11] C. Estan and G. Varghese. New Directions in Traffic Measurement and Accounting: Focusing on the Elephants, Ignoring the Mice. *ACM Transactions on Computer Systems*, 21(3):270–313, 2003.

[12] C. Estan, G. Varghese, and M. Fisk. Bitmap Algorithms for Counting Active Flows on High Speed Links. In *Proc. 3rd ACM SIGCOMM Conference on Internet Measurement (IMC)*, pages 153–166, 2003.

[13] P. Flajolet and G. N. Martin. Probabilistic Counting Algorithms for Data Base Applications. *Journal of Computer and System Sciences*, 31(2):182–209, 1985.

[14] S. Ganguly, M. Garofalakis, R. Rastogi, and K. Sabnani. Streaming Algorithms for Robust, Real-Time Detection of DDoS Attacks. In *Proc. 27th International Conference on Distributed Computing Systems (ICDCS)*, 2007.

[15] P. B. Gibbons and Y. Matias. New Sampling-Based Summary Statistics for Improving Approximate Query Answers. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 331–342, 1998.

[16] F. Giroire. Order Statistics and Estimating Cardinalities of Massive Data Sets. *Discrete Applied Mathematics*, 157(2):406–427, 2009.

[17] P. Indyk and D. Woodruff. Tight Lower Bounds for the Distinct Elements Problem. In *Proc. 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2003.

[18] P. Indyk and D. Woodruff. Optimal Approximations of the Frequency Moments of Data Streams. In *Proc. 37th Annual ACM Symposium on Theory of Computing (STOC)*, pages 202–208, 2005.

[19] B. Kalyanasundaram and G. Schnitger. The Probabilistic Communication Complexity of Set Intersection. *SIAM Journal on Discrete Mathematics*, 5(2):545–557, 1992.

[20] N. Kamiyama, T. Mori, and R. Kawahara. Simple and Adaptive Identification of Superspreaders by Flow Sampling. In *Proc. 26th IEEE Conference on Computer Communications (INFOCOM)*, pages 2481–2485, 2007.

[21] D. M. Kane, J. Nelson, and D. Woodruff. An Optimal Algorithm for the Distinct Elements Problem. In *Proc. 29th ACM SIGMOD Symposium on Principles of Database Systems (PODS)*, pages 41–52, 2010.

[22] E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge University Press, 1997.

[23] G. Manku and R. Motwani. Approximate Frequency Counts Over Data Streams. In *Proc. 28th International Conference on Very Large Data Bases (VLDB)*, pages 346–357, 2002.

[24] S. Muthukrishnan. *Data Streams: Algorithms and Applications*. Foundations and Trends in Theoretical Computer Science, 2005.

[25] A. A. Razborov. On the Distributional Complexity of Disjointness. *Theoretical Computer Science*, 106(2):385–390, 1992.

[26] S. Venkatamaran, D. Song, P. B. Gibbons, and A. Blum. New Streaming Algorithms for Fast Detection of Superspreaders. In *Proc. 12th ISOC Symposium on Network and Distributed Systems Security (NDSS)*, pages 149–166, 2005.

[27] Q. Zhao, A. Kumar, and J. Xu. Joint Data Streaming and Sampling Techniques for Detection of Super Sources and Destinations. In *Proc. 5th ACM SIGCOMM Conference on Internet Measurement (IMC)*, pages 77–90, 2005.