J Netw Syst Manage
DOI 10.1007/s10922-010-9186-5

Uncovering Global Icebergs in Distributed Streams: Results and Implications

Guanyao Huang · Ashwin Lall · Chen-Nee Chuah · Jun Xu

© The Author(s) 2010. This article is published with open access at Springerlink.com

Abstract Discovering icebergs in distributed streams of data is an important problem for a number of applications in networking and databases. While previous work has concentrated on measuring these icebergs in the non-distributed streaming case or in the non-streaming distributed case, we present a general framework that allows for distributed processing across multiple streams of data. We compare several of the state-of-the-art streaming algorithms for estimating local elephants in the individual streams. However, since an iceberg may be hidden by being distributed across many different streams, we add a sampling component to handle such cases. We provide a novel taxonomy of current sketches and perform a thorough analysis of the strengths and weaknesses of each scheme under various QoS metrics, using both real and synthetic Internet trace data. We summarize their performance and discuss the implications for the future design of sketches.

Keywords Iceberg · Sketch · Sampling

G. Huang (✉) · C.-N. Chuah
Department of Electrical and Computer Engineering,
University of California Davis, Davis, CA 95616, USA
e-mail: gyhuang@ucdavis.edu

C.-N. Chuah
e-mail: chuah@ucdavis.edu

A. Lall · J. Xu
Georgia Institute of Technology, College of Computing, 801 Atlantic Drive,
Atlanta, GA 30332-0280, USA
e-mail: alall@cc.gatech.edu

J. Xu
e-mail: jx@cc.gatech.edu

Published online: 24 October 2010

1 Introduction

Detecting icebergs—items whose frequency of occurrence are above a certain threshold—is important for both computer networking and database applications. One example of an iceberg is a network traffic flow (e.g., all the traffic coming from a particular source IP address or targeting a particular destination IP address) that has high aggregate volume across many different monitors, even if it does not appear large at any single monitor. Detecting this type of event is important for a number of applications, including detecting DDoS attacks [1], finding heavy-hitters in Content Delivery Networks [2], discovering worms and other anomalies [3], as well as ensuring SLA compliance [4]. Applications that detect DDoS attacks need to find destination IP addresses that occur frequently across multiple ingress points. To detect SLA violations, an Internet Service Provider (ISP) typically monitors its different ingress links to detect any incoming traffic that violated the stipulated rate limit. In worm detection, the worm signature which appears widely and frequently can be viewed as a global iceberg among all the machines in the network. Global iceberg detection is therefore important for a variety of applications, especially those related to network management and security.

Monitoring and analyzing current Internet traffic is challenging due to increasing link speeds and traffic volume. An offline solution that dumps all the traffic requires prohibitively large space and loses the ability to react in real time. To cope with the high speed and volume, it is only viable to perform low rate sampling [5] or very succinct sketching. The sketches can be constructed using small but high-speed SRAM by either intelligent sampling or data streaming algorithms. Recently, there has been a lot of study on detecting *local* heavy-hitters (also called elephants) in both networking and database scenarios [6, 7] based on sketching. The problem is difficult since there is no prior information about the identities of local heavy-hitters and the sheer volume of the data precludes maintaining per-flow states. The proposed methods include the count-min sketch [6], the count-sketch [8], sticky sampling, lossy counting [9], shared-state sampling (SSS) [10], multistage filters and sample-and-hold [7]. They use hash-based sketches, intelligent sampling, or intricate data structures [11] to preserve the information of local elephants.

However, the aforementioned work only focused on detecting high frequency items (flows) at a single monitor. Since global icebergs may be adversarially split in some scenarios, they are not necessarily local heavy-hitters. It is therefore not enough to only report these local elephants. For example, a DDoS attack might be generated from many botnet zombies such that the traffic volume is moderate at local monitors to evade detection. For these security-related applications we need to accurately measure the aggregated traffic volume from distributed monitors. The detection mechanism should be effective regardless of how the iceberg is split.

In order to detect icebergs, distributed monitors should first measure local traffic for iceberg candidates and then report the measured datasets to a central server, which aggregates count values and extracts the most frequent ones. There are therefore two aspects related to this problem: how to measure local traffic and how to report the datasets. For the first aspect, it is important to detect not only local heavy-hitters but also the split global icebergs. For the second aspect, since

available bandwidth is considerably less than the sheer volume of data, it is necessary to reduce the communication cost.

We identify three QoS metrics for the identification of icebergs: false positive rate, false negative rate, and the average relative error of the size of the icebergs. Besides these, the communication cost (between the distributed monitors and the central server) should be reduced to orders of magnitude smaller than in the naive solution in which every local flow is stored and reported. We must also ensure that the local memory requirement of our solution is reasonable, given the limits on SRAM.

We approach the global iceberg detection problem using a combination of local sketching and uniform sampling at each distributed monitor. We will demonstrate that both these components are necessary for accurate detection of global icebergs. Uniform sampling across all the monitors significantly improves the likelihood that the split icebergs will also get detected. The local sketches are necessary for obtaining accurate estimates for the counts of the icebergs. We perform a comprehensive study on the combinations, showing how our solutions perform for each of the performance metrics. Our contributions are as follows:

- First, we introduce the new problem of detecting distributed icebergs in the streaming setting, i.e., with non-zero local measurement errors. We propose a combined sketching and sampling approach for detecting and measuring them. While sketches are useful in detecting local elephants, we verify the importance of uniform sampling in capturing distributed global icebergs.
- Second, we perform a thorough evaluation of the solution space for the sampling and sketching methodology described above. We develop a taxonomy of different sketches based on the techniques they use. According to this taxonomy, we develop and compare two main strategies that combine sketching and uniform sampling in local measurement using real network data. Our comparisons are conducted using both real and synthetically-generated data sets.
- Third, we modify an existing algorithm to improve global iceberg detection in all the aforementioned metrics. Although this modification is not optimal for detecting local elephants, we demonstrate its efficacy in detecting global icebergs when it is combined with uniform sampling. This study helps understand the difference between local elephant detection and global iceberg detection.
- Finally, we compare the performance of existing sketches for different split patterns of global icebergs. This helps in understanding the relationship between detection efficiency and the underlying traffic pattern. We show that our methodology is robust against several split patterns that may be used by an adversary to hide an attack.

This paper extends our previous work [12]. We add in more detailed discussion of the different sketches and the ways they can be combined with sampling. We conduct experiments on more traces, and present sensitivity analysis. The paper also analyzes the communication cost associated with our different combinations of sketching and sampling solutions. The implications of our results for future design of sketches are included in the discussion section.

The rest of the paper is structured as follows. Section 2 discusses related work. In Sect. 3 we formally define our problem. We present our solution in Sect. 4 and also describe the sketches that we use and a taxonomy via which we can compare their properties. Section 5 presents our experimental study on comparing different sketches as well as different combinations. Both real Internet traffic traces and synthetic data are used. Our results and observations are discussed in Sect. 6. Contributions and results are summarized in Sect. 7.

2 Related Work

While there has been considerable work that focuses on identifying heavy-hitters at a single node [6–9, 11], we believe that detecting global icebergs is a far more interesting and challenging problem. Streaming processing techniques have been closely studied in the theoretical setting [13–15] for various purposes. However, our goal is to come up with practical algorithms for both detecting and estimating the size of icebergs in real data sets. Recently, Cormode et al. [16] proposed the problem of functional monitoring, where the local nodes continuously send updates only insofar as needed to satisfy some global constraint (e.g., detecting all the icebergs). Our work differs from theirs since we assume fixed measurement periods, which potentially allows us to have more communication-efficient mechanisms. Manjhi et al. [17] studied the problem of discovering icebergs in a distributed environment when nodes are arranged in a multi-level communication hierarchy. We study the simpler, practically motivated single-level communication scheme instead. Additionally, their scheme is dependent on a global iceberg being frequent in one or more local sites We do away with this assumption by leveraging the sampling component of our algorithm to detect, with high probability, global icebergs that may not have a high count at any site.

An alternate way of defining an iceberg is to consider top-k queries rather than fixing a threshold [18]. This is studied by Babcock et al. [18] in the distributed setting, and extended by Olston et al. [19] to support sum and average queries. These approaches aim to keep the local elephants aligned with the global ones and hence face the same issue as the above solution [17]—icebergs that are finely distributed among the local nodes are hard to discover.

In [20], Zhao et al. propose two methods for discovering global icebergs in distributed data. Their first scheme involves size-based sampling where they derive the optimal sampling rate for local sites. In their second scheme, they use Bloom filters to summarize the information at local nodes, and demonstrate a quantization scheme that is independent of the manner in which the iceberg may be split. Our goal is to also design split-independent algorithms. However, the main difference of our work from [20] is that we study the much more challenging streaming version of the problem, where we cannot assume that the local frequencies are known exactly at the local sites.

Recently, [21] proposed a sketch to summarize local streams and provide unbiased estimators for subpopulation sizes. This sketch retains more information

(i.e., distributions) than we require and hence is more complicated and expensive. Cormode and Hadjieleftheriou [22] compared the different sketches in detecting local elephants. We differ from their work since we focus on a more challenging problem: detecting icebergs from distributed streams. The goal of our work is to find a practical and efficient solution for this problem.

3 Formulation

Let us assume that there are n monitors, each with comparable numbers of items. At each of the n distributed points there are streams of items presented as update pairs $\langle \mathrm{id}, c \rangle$, where id is the identity and c is the update value for this pair. An item may appear many times at the same monitor and also at multiple different monitors. The *global count* for an item is defined as the aggregate count value of all pairs which belong to this item across all the monitors. We denote by S the sum of the global counts of all the items. An iceberg is defined to be any item whose count aggregated across all n points is at least $T = \theta S$, for some $\theta \in (0, 1)$.

Our primary goal is to detect all such icebergs. Secondly, for each iceberg detected, we want to estimate its aggregate size accurately. Finally, we would like to achieve all this while keeping the false positive rate as low as possible. It may be the case that our methods will erroneously count items with counts slightly less than the threshold as icebergs. Since this is unavoidable, we introduce a parameter $\omega < \theta$ and say that we will not count as a false positive any item whose count lies within $(\omega S, \theta S)$. For many security applications, it is more important to not miss an iceberg than it is to not report items smaller than the iceberg threshold.

We define I to be the set of items whose global count values are larger than T. Then, our goal is to find an estimation set I^* as well as their estimated count values. The solution should satisfy the following requirements:

- The estimated set I^* should have few to no false negatives and false positives. Meanwhile, the estimated count values of items in $I \bigcap I^*$ should be close to the real count values of these items.
- The communication cost from the n nodes to the central server should be considerably smaller than the cost when every node aggregates and reports all their counts exactly.

In order to find I^*, every monitor first measures and generates a local data set which is a summary of the local traffic. The nodes then send the distributed data sets to the central server. We call these *estimated distributed streams* because exact summarization of the local streams is impossible due to resource constraints at the local nodes.

Note that, for traffic data, we may be interested in either the total number of bytes that comprise a flow or simply in the total number of packets (i.e., every update is of the form $\langle \mathrm{id}, 1 \rangle$ in our above notation). Our solution is for the latter case, though we note that it can be easily extended to apply to the former case as well.

4 Sketching and Sampling

By understanding the capability of both uniform sampling and sketching, we propose efficient combinations of them to attack the global iceberg detection problem. The combination works as follows. Every monitor first uses uniform sampling and a sketch to summarize its traffic. We denote by LS the items that are captured locally by uniform sampling. Similarly, LH denotes the set captured by the local sketch, i.e., the local heavy-hitters. We exclude items in LS which are already present in LH since the sketch is better at estimating the count values of local elephants. Define SH = LS − LH to be the set of items that are found exclusively by sampling. Finally, the distributed monitors send the LH and SH lists as well as their estimated count values to the central server, which aggregates these estimates to get the global icebergs. This methodology is illustrated in Fig. 1 and the rationale behind it will be explained in the following section.

4.1 Advantages of Sketching and Sampling

In this section we summarize the properties of sketching and uniform sampling to demonstrate the importance of combining both of them in capturing global icebergs. The sketches we study include count-min [6], count-sketch [8], sticky sampling, lossy counting [9], SSS [10], sample-and-hold, and multistage filters [7]. The sketch in [11] can only extract the top k most frequent items without estimating their values in one pass. Since our goal includes estimating the size, we do not consider this sketch further. Next, we briefly describe each of these sketches; for a detailed description please refer to the respective papers.

Among the sketches we compare, sticky sampling, lossy counting, and sample-and-hold continuously keep track of current large flows.

The sample-and-hold [7] sketch samples every incoming packet with some fixed probability. Once a flow gets sampled, its later packets will keep being captured.

Sticky sampling [9] improves sample-and-hold by periodically excluding small and medium flows from the sketch. Even if one flow was sampled, it might be removed once its size is not large enough.

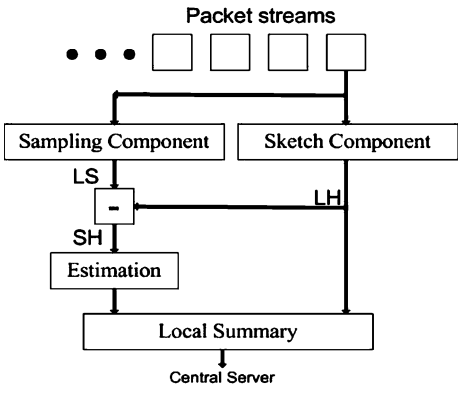


Fig. 1 Our solution

Lossy counting [9] goes further in that it can estimate the past sizes of a flow. Note that a flow that initially has low frequency might not have a record created for it. After this flow becomes large, it will be estimated as an elephant; the previously unrecorded packets for it should therefore also be estimated. Lossy counting maintains a pair $\langle f, \delta \rangle$ for every item, where f stands for its approximate count value after it is inserted into the sketch and δ is the maximum value by which f is under-counted. Items will also be continuously deleted from the sketch if their $f + \delta$ values are not large enough.

Different from the sketches above, SSS, multistage, count-min and count-sketch use arrays of counters to maintain the information of every flow. Every packet will update every array of counters, the locations of which depends on hash functions applied to the flow identity. An elephant is detected if the estimated size from the counters is above a threshold.

In multistage [7], every packet encountered in the stream is used to increment a set of counters (indexed using hashing) by the packet size, and the estimated size is the smallest size of all the corresponding counters in the sketch. If the estimated size is larger than a threshold, the flow is retained as an elephant and inserted into flow memory.

Count-min [6] shares similar algorithm and architecture with multistage. However, count-min uses heap to store the large items, while multistage uses hardware hash tables.

Different from the above two sketches, count-sketch [8] estimates the sizes in a different way. In count-sketch, an incoming item might increase or decrease the counters, based on the results of hash functions. The estimation of the item size is no longer the smallest of all the associated counters, but the median of them.

SSS [10] can be viewed as an improvement of multistage. In SSS, a packet will pass a sampling component before it updates the counters. The sampling component helps discard small flows.

We make the following observations on the seven sketches above:

Observation 1 *In all seven of the sketches that we consider, the estimation error for an item (flow) is independent of its real count value.*

For sketches that maintain counts using hash functions, the estimation error is caused by hash collision which is independent of the real sizes. The error of sample-and-hold is introduced by sampling the initial packets before the item is first sampled, which depends only on the sampling probability. In sticky sampling, the error is introduced by missing the first packets as well as randomly deleting packets from the sketch, which are both independent of the real count value. In lossy counting, the error is introduced by estimating the count values of the item before it is inserted into the sketch, which is related to the arrival pattern instead of the size. Please refer to the original papers for details.

Therefore, the estimation error for one item can be viewed as a function of traffic arrival pattern and sketch setting, instead of a function of its own count value. For instance, the error introduced by count-min is bounded by a constant multiplied by the first order norm of the total traffic volume [6], which is relatively small

compared to large items; however, for small items, it might be larger than the error introduced by sampling.

Observation 2 *The estimation variance by reverting uniform sampling is an increasing function of the true count value.*

The second observation can be easily proved assuming independence of the samples. The reason why we combine sketching and uniform sampling is therefore obvious. First, sketches generally sacrifice the estimation accuracy of small- to medium-size items in order to preserve information for large items. They are insufficient since the global iceberg can be split and appear as non-elephant flows locally. Second, even if the global iceberg is split, the probability that this item is sampled by uniform sampling remains large. This suggests that we can use sketches to capture local heavy-hitters and use uniform sampling to capture local non-elephants which might be global icebergs.

4.2 Why Sampling and Sketching are Both Necessary

Based upon the observations above, we further justify our combination by analytically studying their different roles in detecting global icebergs. First, we show that local sketches can assist us in detecting global icebergs. Second, we show that sampling helps us estimate their counts when they are hidden as non-elephants locally. Namely, the sketching can capture iceberg identities while the uniform sampling helps more accurately estimate their sizes.

Recall that our goal is to find all icebergs with aggregate frequency of at least θS, where S is the total packet count across all n monitors. Let us denote by $s_1, s_2, \ldots, s_n$ the packet counts at each of the monitors. We fix a single global iceberg with count $c \geq \theta S$ that is distributed across the monitors with counts $c_1, c_2, \ldots, c_n$. Suppose that the local elephant detection sketches can successfully identify all items that comprise at least θ' of the local traffic. Let us assume, without loss of generality, that our item of interest does not appear as a local elephant at monitors $1, \ldots, k$, i.e., for each $i \in \{1, \ldots, k\}$, $c_i \leq \theta' s_i$. Then, at all the other monitors, this item appears with count at least

$$\sum_{i=k+1}^{n} c_i = c - \sum_{i=1}^{k} c_i \geq \theta S - \sum_{i=1}^{k} c_i$$
$$\geq \theta S - \sum_{i=1}^{k} \theta' s_i \geq \theta S - \theta' S.$$

Hence, assuming that the local sketches detect all the θ' local elephants and report their full counts, at least $(\theta - \theta')S$ of the iceberg's count gets reported to the central server. Now, if we choose $\theta' = \theta - \omega$, we get that at least ωS of the iceberg count gets reported to the central server, which can safely declare the item an iceberg.

There are two drawbacks to the above analysis. First, sketches are not guaranteed to return the θ' local elephants; they may miss some with some small probability. Second, while some local monitors successfully report the iceberg identity, its

aggregated size may be severely underestimated since parts of it are not reported. Both these issues can be resolved by the sampling component. Since packets are sampled uniformly at all the measurement points, the results of sampling are independent of the manner in which the global iceberg is split between them. In the case where the global iceberg is a not a local elephant, uniform sampling will still identify many samples of it; the expectation of the sampled flow size is linear with the actual flow size.

Now, one may ask why we did not rely exclusively on uniform sampling. This can be answered by studying the estimation error for reverting sampling. With sampling rate p, an item of count c has estimation variance $c(1 - p)/p$ [23]. In comparison, the sketches that we are using have considerably less error. The combination therefore has smaller estimation error than exclusively using uniform sampling.

Therefore, leveraging the synergy of local elephant detection sketches and uniform sampling is ideal for this problem.

4.3 Taxonomy of Sketches and Combinations

In this section we present different combinations of uniform sampling and sketching. We categorize the combinations based on a novel taxonomy of the seven sketches mentioned earlier. The sketches can be divided into two main categories:

- **Implicit Counters (IC)**, in which an identity can be used to query counters for an estimate. SSS, multistage, count-min, and count-sketch belong to this category.
- **Explicit Sketches (ES)**, which includes sample-and-hold, sticky sampling, and lossy counting. The sketch itself does not include any counter arrays.

SSS, multistage, count-min, and count-sketch can be viewed as multiple stages of counting Bloom filters, where multiple stages are used to reduce hash collisions. Since the arrays of counters store lossy information of all the underlying traffic, they can be used to estimate the size of an identity. In contrast, sample-and-hold, sticky sampling, and lossy counting have no such counters. If one sampled item is not contained in the sketch (estimated as potential elephant), its count value can only be estimated from the uniform sampling component through our combination of sketch and sampling.

Based on the taxonomy of the sketches, there are two main combinations of sampling and sketching:

- **RS (Revert sampling + sampling)**: Use a sketch to detect and report local elephants as well as sampled non-elephants to the central server. The count values for sampled non-elephants are estimated by reverting the sampling rate. This combination works for the seven sketches.
- **QC: (Query counter + sampling)**: For the implicit counters, the count value for local non-elephants can be estimated by querying the counters. Distributed monitors may therefore report the counters, and use the sampled identities to query the collected counters. This combination works for the four sketches belonging to Implicit Counters.

The combinations correspond to different methods in estimating sizes of items in SH in Fig. 1. For QC, local monitors report the local elephants and sampled non-iceberg identities as well as their count values. They do not send the entire counters. Measurement points locally estimate count values for non-icebergs by querying the counter. In our detailed study [24] we also analyze other variations for QC.

We next show the results of comparative studies where we tested our methodology, including the various RS and QC combinations, on both real and synthetic data.

5 Evaluation

In this section we evaluate our approach through experiments performed on real and synthetic Internet traffic data. We compare the performance of different sketches to find the the best one that can be combined with uniform sampling to give the best estimate of the set (size) of global icebergs. We also use synthetic data to study how the split of the global iceberg influences the detection accuracy.

We compare the seven sketches as well as the naive sampling and naive iceberg approach.

1. **Naive Sampling Approach**: uniform sampling alone is used and count values are estimated by reverting sampling. There is no local sketch to detect local elephants.
2. **Naive Iceberg Approach**: only local heavy-hitters are collected by the sketch and reported to the central server. There is no sampling component.

We show that our combination outperforms these naive approaches for comparable communication costs. Besides the comparison of different sketches and combinations, we improve multistage by a slight modification of the original algorithm. This modification helps us understand the capability of combining sampling and sketching in global iceberg detection.

Internet traffic data serves as a good candidate to test our methodology since there are several practical applications for measuring globally distributed icebergs, e.g., detecting DDoS attacks, SLA measurements, and detecting Internet worm proliferation. The item identities over which we can detect global icebergs can be any subset of the standard IP five-tuple (⟨srcIP, dstIP, srcPort, dstPort, protocol⟩). We choose the destination address for our experiments because of its usefulness (e.g., for DDoS attack detection), though our methods will work equally well for any other combination.

5.1 Experiment Settings

5.1.1 Data Trace

We use two traces from Abilene [25] network collected at different times. There are 11 and 9 sites for the two traces respectively; Abilene network reduced 2 sites after the year 2009. This helps us evaluate the performances under different number of

monitors. The records are sampled data with sampling interval 100, i.e., 1 out of every 100 packets get sampled. We revert this sampled data to construct the original data. Since sampled records miss many of the smaller flows, we generate and insert small flows such that 10% of the flows contribute about 80% of all the traffic. This emulates heavy-tail flow size distribution [26]. We introduce $9 \times N$ small flows where N is the number of iceberg flows contained in the traces. The size of introduced small flow follows a uniform distribution with an aggregated size about 20% of the total traffic. This simple conversion does not guarantee the precise percentage. However, it is enough to showcase the performance under heavy-tail flow size distribution. These small flows are not icebergs. However, they will effect the accuracy of iceberg detection. We use the time stamps of flows to determine the ordering of the items in each stream, assuming that interleaving flows that were temporally close to each other did not significantly affect the performance of the sketches. For instance, the number of concurrent flows is small; the probability that they are hashed into the same counter in multistage is also small. Therefore, the performance should not be much influenced even without packet level traces. Note that these traffic reverting procedures do not affect our set of global icebergs, though they do affect the performance of our sketching and sampling components.

5.1.2 Iceberg Parameters

In our experiments, we attempt to detect icebergs whose sizes are larger than $\theta = 0.1\%$ and $\theta = 0.01\%$ of the total number of packets across all the monitors. As we mentioned earlier, it is difficult to distinguish items whose counts are slightly below the threshold from those above the threshold, so we introduce a parameter ω that is the ratio above which we do not penalize the detection algorithm for detecting a false positive. In all our experiments we set $\omega = 2\theta/3$, i.e., two thirds of the iceberg threshold.

5.1.3 Metrics Examined

In our experiments, we measure the following three metrics in diminishing order of importance. First, we are concerned with the probability that we miss an iceberg. It is of paramount importance that an anomaly is detected in most applications. Second, we measure the average relative error of the detected icebergs. We define relative error in the standard way: the absolute error divided by the true size. Accurate estimation of the sizes of the icebergs is important for gauging the magnitude of the problem. Finally, we aim to minimize the false positive rate. This is also important because a false positive will cause resources to be unnecessarily wasted in responding to a false alarm. Besides these three metrics, we also perform sensitivity analysis with respect to different traces and different SRAM memory sizes.

Another metric we should compare is the average number of operations per packet. Since we intend to process all the packets at line speed, we must use fast SRAM which allows a few operations per packet. We will come to this point at the end of the comparisons.

5.1.4 Settings of Sampling and Sketches

We vary the uniform sampling interval from 100 to 30,000 to study the influence of different sampling intervals. In our data, the number of distinct flows is about 70,000−90,000 at every point. For Implicit Counters, we vary the counter settings from 4 × 5,000 (4 stages each with 5,000 counters) to 5 × 6,000, since these sizes can yield better results than other settings [24]. We chose this range of sizes from our exhaustive experimental study. Estan and Varghese[7] measures in continuous intervals of 5 s, while our experiments use traces of 5 min.

All our local sketches use the same amount of memory and hence have identical parameters. We set the local SRAM size to be 1 Mbit, which is about 5,000 entries for local elephants, as suggested by [7]. In Implicit Counters, the counter occupies additional space and thus can only store fewer records. For instance, when the counter size is 5 × 5,000, there are only about $(5{,}000 \times 32 - 5 \times 5{,}000 \times 4)/32 = 1{,}875$ (32 bytes are used for one entry [7]) entries available to store elephants. In our experiment we try different parameters to satisfy the memory requirement. Note that it is usually difficult to guarantee the exact amount of memory used by the sketch. Therefore we also experiment with different memory sizes. This helps us understand the robustness of the sketches.

5.2 Comparison Roadmap

In these experiments, we first compare the seven sketches, using RS to estimate sizes of non-local-elephants. We find that SSS and multistage are generally the best sketches. This result is also confirmed when comparing four IC sketches using the QC combination. We then compare both QC and RS combinations with naive approaches to find the best combination and the best sketch. Based on the observations we slightly modify multistage to study the difference between global iceberg and local elephant detection. Finally, we use synthetic data to examine our best sketch as well as to study the influence of different split patterns on global iceberg detection.

5.3 Comparison for RS

In this section, we first compare different counter sizes for Implicit Counters to find the best counter size setting. We found that the best counter size is 4 × 7,000. We use this counter size to compare the four sketches (Implicit Counters) with sticky sampling, lossy counting, and sample-and-hold. The comparison of different sketches are illustrated in Figs. 2 and 3. For all the experiments below, the graphs for $\theta = 0.1\%$ have similar trends as $\theta = 0.01\%$, therefore we omit the graphs for $\theta = 0.1\%$ and focus on the smaller threshold. Details can be found in [24].

SSS and multistage have the best results in almost all the metrics. Sample-and-hold and sticky sampling always show large probability of false negative while countmin and countsketch show large average relative error. Lossy counting has a large probability of false positive. Figures 2 and 3 suggest a larger sampling interval will always deteriorate the overall performance. When the sampling interval becomes larger, inaccuracy introduced by sampling dominates.

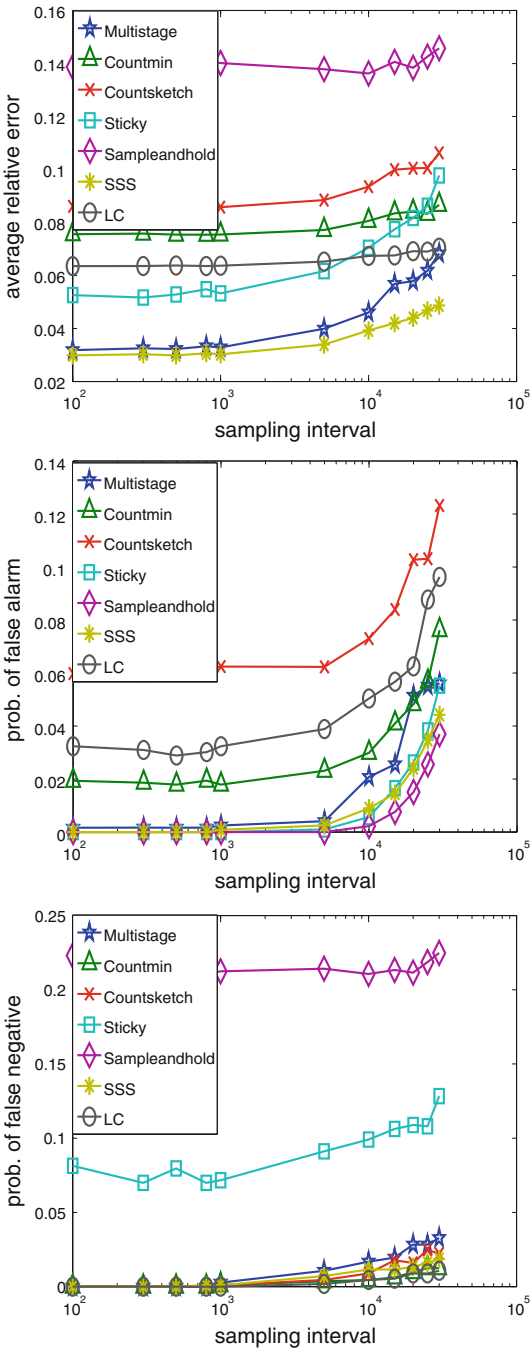


Fig. 2 Comparison of sketches in RS, trace 1 ($\theta = 0.01\%$)

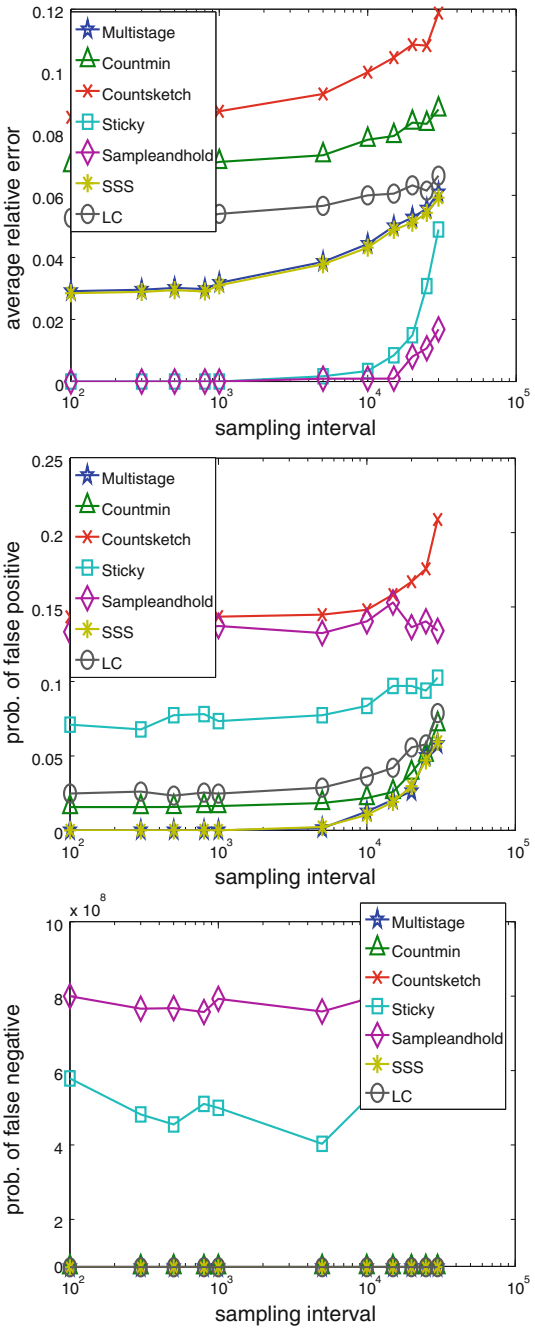


Fig. 3 Comparison of sketches in RS, trace 2 ($\theta = 0.01\%$)

We also evaluate their performance by changing the memory size of SRAM. In Fig. 4, the x-coordinate is the number of bytes used by SRAM divided by 32 (32 byte per entry). For Implicit Counters this number also includes the space for

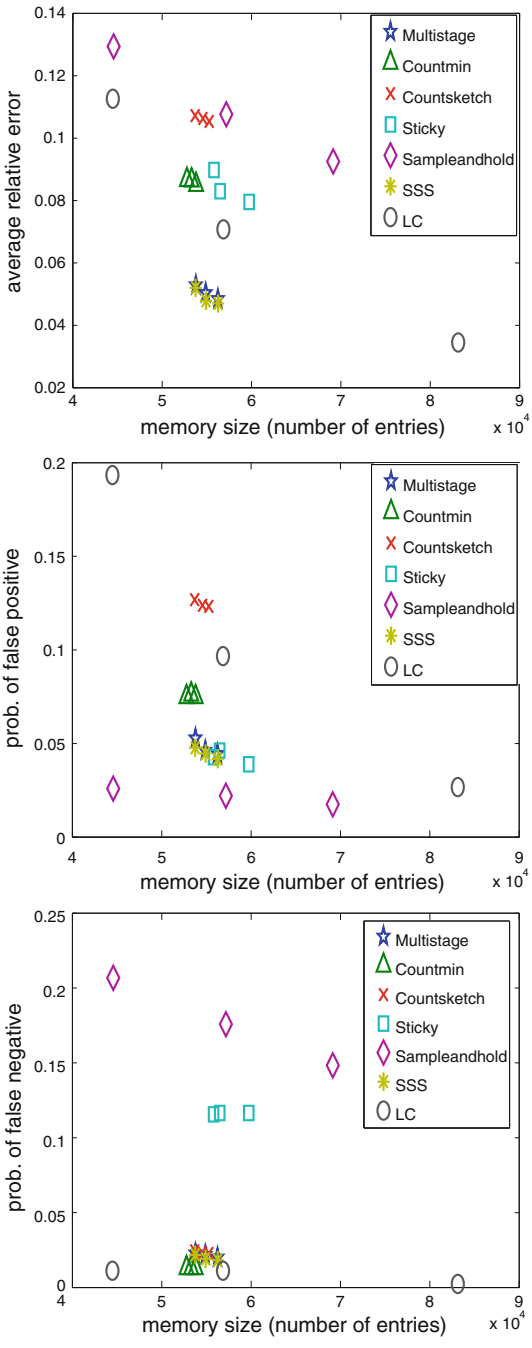


Fig. 4 Sketches in RS under different memory size, trace 1 ($\theta = 0.01\%$)

counters. We only plot the results when the uniform sampling interval is 30,000. As discussed later, only at this large sampling interval can the communication cost be reduced to an acceptable level. Results suggest that the performances are stable; SSS and multistage remain the best sketches.

We have the following four propositions to explain the results.

Proposition 1 *Multistage is better than count-min in terms of probability of false alarms. This is because multistage uses hash table in the flow memory to keep the estimated elephants. When a new packet comes in, multistage first checks whether this packet is already contained in flow memory. It therefore has the opportunity to directly update the flow memory, instead of increasing hash collisions in the counters* [7]. *Instead, countmin always uses packets to update counters. With every packet entangled in the counters, a small or medium item is more likely to be misreported as an elephant.*

Proposition 2 *Sticky sampling is a combination of sample-and-hold and the KSP algorithm. Sticky sampling acts stickily: an item's record will keep being updated once it is sampled. However, the sampling rate is decreasing as more packets are processed. This makes it similar with KSP, i.e, decrease count values to delete the smallest existing elephant. The deletion of smallest elephant is random, which is different from KSP. The combination of sample-and-hold and KSP makes it better than sample-and-hold in most cases. Sample-and-hold is inferior since it never deletes items from the sketch. An initially burst flow (which is not elephant) will occupy the sketch without vacating locations for the later elephants.*

Proposition 3 *SSS is better than multistage since it is more selective in items with size around the threshold* [10]. *This is achieved by the sampling component (filter) before arrays of counters. An item can only update the counters after it passes the sampling component, which helps to discard small flows. The sampling component therefore reduces both the probability of false negative and probability of false positive. We also notice that their performances are very close to each other.*

Proposition 4 *Lossy counting has large probability of false positive, but extremely small probability of false negative. In lossy counting, the count value of an item before it is inserted into the sketch will also get estimated, which is the maximum value it could have appeared* [9]. *Therefore, lossy counting actually introduces large overestimation.*

Another interesting observation is that countsketch actually has large probability of false positive and average relative error. Although it was designed to be an unbiased estimator, its experimental results are unpromising.

5.4 Comparisons for QC

In this section, we compare Implicit Counters using QC. Similar with Sect. 5.3, the best counter size is still $4 \times 7{,}000$. From the results in Fig. 5, we see that still SSS and multistage generally have smaller average relative error and probability of false alarm than other sketches, but larger probability of false negative than count-min,

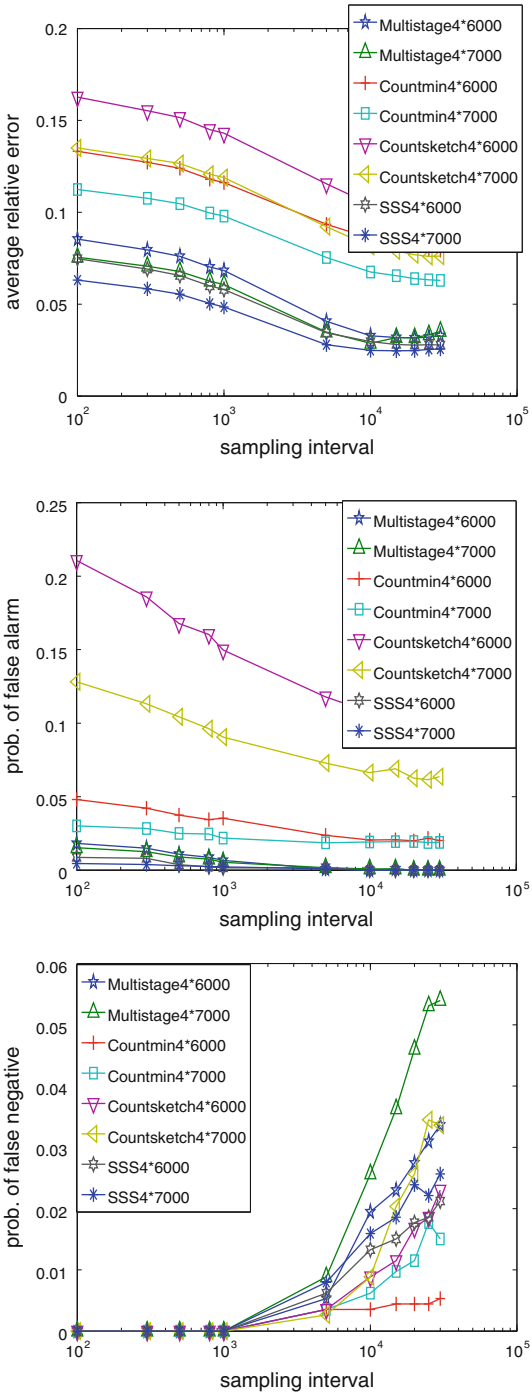


Fig. 5 Comparison of implicit counters in QC, trace 1 ($\theta = 0.01\%$)

for most counter size settings. We omit the results for the second trace since it is very close to Fig. 5.

The trend of the figures are due to hash collisions in the counters; the count values are estimated by using the sampled identities to query the counter. Since smaller sampling rates cause more items to be missed, the average relative error and probability of false positives improve with smaller sampling rates: the effects of missing items and overestimating count values cancel each other out. However, the probability of false negative deteriorates with larger sampling interval.

5.5 Comparing QC and RS

In this section we compare QC and RS. We omit count-sketch and count-min since they are consistently worse than multistage and SSS in previous comparisons. We omit sticky sampling and sample-and-hold since they show large probability of false negative. Although lossy counting seems promising in probability of false negative, its probability of false positive is high, and it require extremely large number of operations per packet. We only present results for counter size $4 \times 7{,}000$. Meanwhile, we also present the results for sketches using the naive iceberg approach as well as the naive sampling approach. This helps us understand the limitations of the naive approaches. The results are illustrated in Fig. 6. The results for the second trace is similar and omitted here. Here "NaiveMultistage" ("NaiveSSS") simply means naive iceberg approach with local multistage (SSS) sketch.

The communication cost is the number of items that will be sent to the central server. In our experiment, each reported item uses 32 bits (16 for item identity and 16 for its size). In both RS and QC, suspected icebergs are reported to central server along with their sizes. We find out that the cost mostly depends on the sampling interval and remains similar for different sketches under same combinations. This is because most of the randomly sampled identities are non-elephants. Those non-elephants usually dominate the reported items. Since we use the similar memory for all the sketches, we only plot the communication cost for multistage in Fig. 7. For the intervals we examined, the total costs are around 25 MB for sampling interval 100 and 580 KB for interval 30,000. The naive communication cost is roughly 64 MB, in which case every local packet is dumped and every item is reported. For the naive iceberg approaches, the communication cost is irrelevant to the sampling interval, and remains constantly smaller than other combinations. The combinations generally utilize more memory to capture the split icebergs. When sampling interval is 30,000, the number of entries reported by the sampling component is around 10% of the number of entries in SRAM maintained in the sketch, which suggests we use 10% more memory to detect split icebergs [tbp].

There are a few points worth noting:

1. The naive sampling approach is accurate when the sampling interval is small. This does not conflict with [7], since we are measuring the number of packets instead of the number of bytes. For the latter case, the variation of individual packet size introduces more error. The inferior performance of our combination

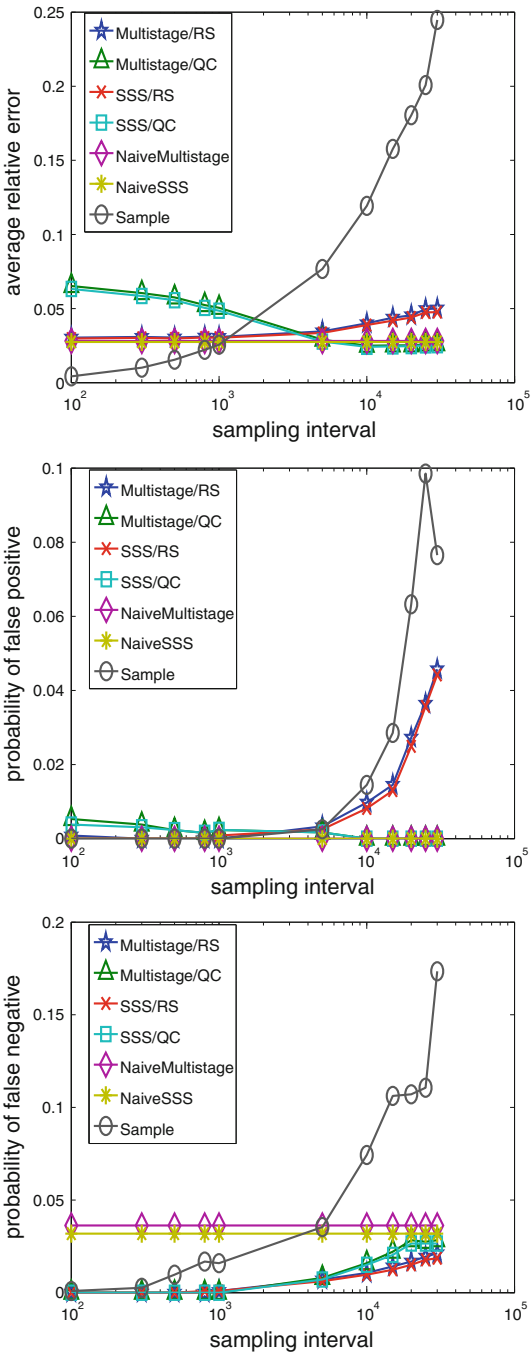


Fig. 6 Comparison of sketches in QC and RS, trace 1 ($\theta = 0.01\%$)

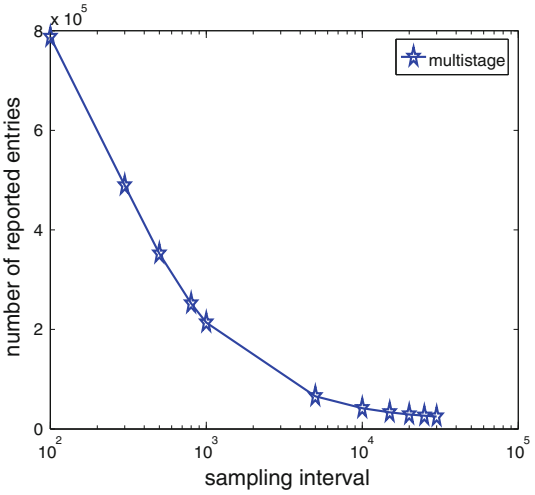


Fig. 7 Communication cost for multistage. Combination RS. Trace 1

also suggests that there are large hash collisions in one measurement interval of 5 min. However, naive sampling is very sensitive to the sampling interval; even when it is as large as 100, the total communication cost is still prohibitively large.

2. When sampling interval is small, RS is better. When the interval grows, QC becomes better. Note that only when the sampling interval is around 30,000 can the communication cost be reduced to 580 KB/64 MB $\approx$ 1% of the naive approach. QC is more feasible since we also aim to reduce the communication cost.
3. The naive iceberg approach has similar average relative error and probability of false alarm as RS and QC in large sampling intervals. However, it has larger probability of false negatives. We will also show its inferiority in Sect. 5.7 .

The results suggest that QC is the most efficient combination. For small communication cost, SSS and multistage perform better in QC rather than RS, in terms of average relative error and probability of false alarm. However, RS has a smaller probability of false negatives. This suggests that we can use the sampled identities to query the local counters for estimation. This can yield good performance even if the uniform sampling interval is large. The total communication cost can be reduced to 1% $\approx$ 0.58/64 of the naive approach, i.e., two orders of magnitude improvement.

5.6 Improving Multistage

In this section we slightly modify the original multistage algorithm. In our modified multistage, once an elephant is reported from the counters to the flow memory, we decrease the counters by the estimated count value. This is slightly different from the original multistage in [7], in which it only refreshes the counters at the

beginning of each measurement interval. Intuitively, the large flows which are deleted from the counters and inserted into the flow memory (a hardware hash table) will not influence any other future items. Note that the removal process will not influence the reported elephants, since the estimate remains the same as it is in original multistage algorithm. However, this process might underestimate other items. This is because the decreased estimate might be larger than the true elephant size (due to possible hash collision). The values of other items who happened to be hashed into the same location thus might get underestimated. Interestingly, we find the overall performance is better than the original multistage algorithm in every aspect.

We come to this modified multistage because of two reasons. First, overestimation does not always decrease the false negative probability. As we have shown in Sects. 5.3 and 5.4, in some settings, the overestimation might cause non-icebergs to mask real icebergs, which increases the overall probability of false negatives. Secondly, in distributed global iceberg detection, we use sampling to complement the sketch. Even if the sketch underestimates some items at one monitor, their values can be estimated by the sampling component if they are global icebergs.

The results of the overall performance for this modified algorithm are in Fig. 8. "MMultistage" in the figures denotes our modified algorithm. "MMultistage" has smaller average relative error, probability of false alarm and probability of false negative than multistage for both QC and RS. It is even better than SSS for most metrics.

By this modification, large flows have less impact on the small ones (since counters are also decreased); fewer small flows will be misreported as elephants. Therefore it decreases the elephant threshold for the same memory size: we can maintain more elephants in the sketch. For example, suppose, initially, a flow will be inserted into the flow memory once its estimated size is larger than 10000; now, this value can be reduced to 8000 to allow more items. The earlier a flow is inserted into flow memory, the more accurately it will be kept. To demonstrate how the local detection influences global detection, we especially look into two split pattern of global iceberg whose size is slightly above $0.01\%S$ [tp].

- The iceberg is split uniformly. It appears with frequency slightly above the threshold at every monitor. Since sampling can complement sketching, the overestimation by hash collision in counters and the underestimation introduced by the modified multistage have opportunities to cancel each other out.
- The iceberg is split following a Zipfian distribution. It appears much larger than the threshold in some places and as small flows everywhere else. For the top local heavy-hitters, the estimation is more accurate since they are inserted into flow memory earlier. The small flows are estimated by reverting sampling; the estimation remains the same for different detection schemes.

Meanwhile, we find that this modification increases the probability of false negative when detecting local elephants in Table 1. Sampling does not necessarily complement sketching in local elephant detection, but does help capture the underestimated values in global iceberg detection.

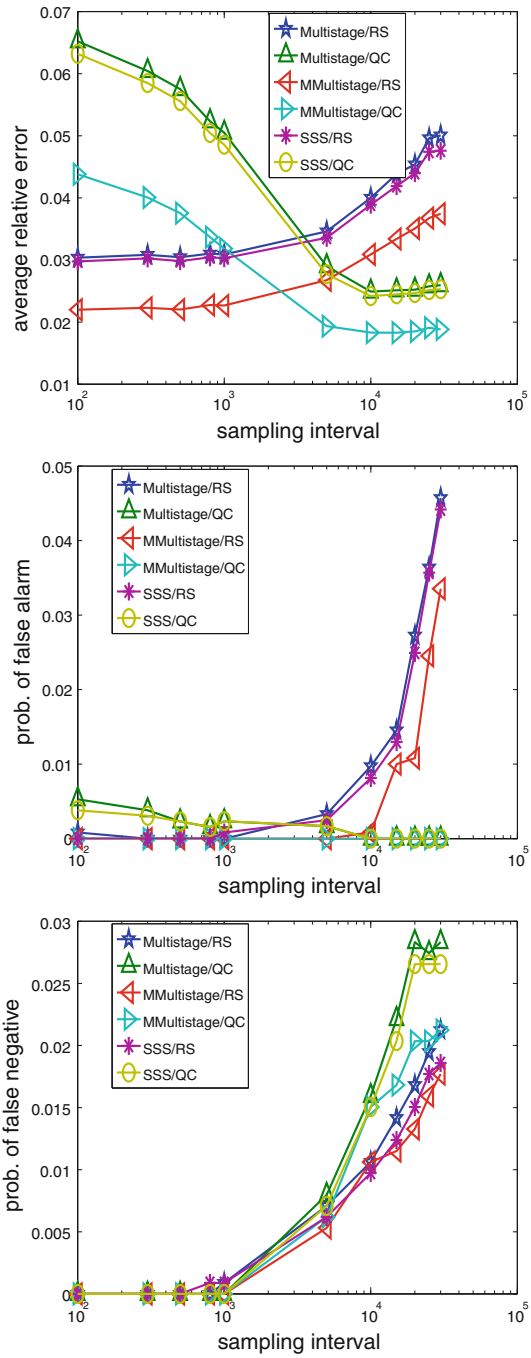
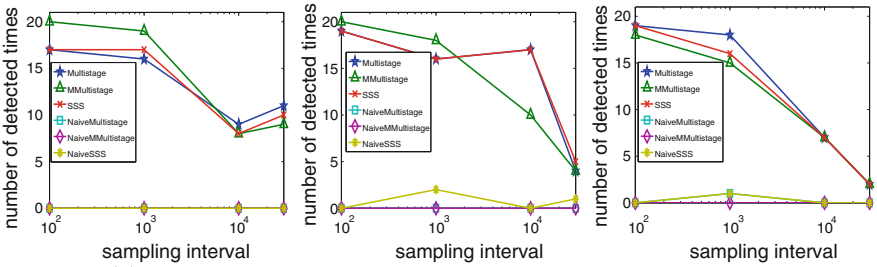


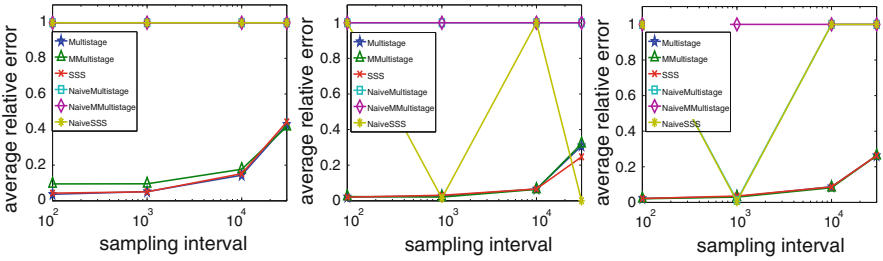
Fig. 8 Modified multistage in Combination QC and RS in real data, trace 1 ($\theta = 0.01\%$)

Table 1 Modified multistage in local elephant detection $\theta = 0.01\%$

	Multistage	MMultistage
Average relative error	0.075725	0.047467
Probability of false alarm	0.141296	0.009615
Probability of false negative	0.000000	0.003391



(a) Number of times detected for uniform, gaussian, and zipfian split patterns



(b) Average relative error for uniform, gaussian and zipfian splits patterns

Fig. 9 Detection accuracy for one inserted iceberg in synthetic experiments

5.7 Experiments on Synthetic Data

In this section we focus on the detection accuracy of one particular inserted iceberg whose size is slightly above the threshold. We also study how the detection accuracy will be influenced by different split patterns of this iceberg. We only use combination QC and compare the results of multistage, SSS, and MMultistage since they have shown promising results. The inserted iceberg size is 93,884, while the total packet number (of all flows) is 93883075 ($\theta = 0.1\%$). We split the inserted iceberg in different nodes ($n = 20$) according to uniform, gaussian, and Zipfian distributions. Every experiment is repeated 20 times. The results are in Fig. 9. More results on the performance of different sketches on synthetic data can be found in [12]. Our findings are as follows.

1. The naive iceberg approach is infeasible for all the cases, even if the iceberg is split non-uniformly. The worst case for naive iceberg approach is uniform split: none of the sketches manage to detect the iceberg. When iceberg size is small,

even if it is large at some monitors, its overall size will be underestimated because some count values are not included. When the split is non-uniform, only naive SSS and naive multistage managed to detect the iceberg once.

2. When using QC, the gaussian and Zipfian split both have better results than uniform split in terms of average relative error. However, the detection accuracy is not necessarily better in gaussian and Zipfian split. This suggests that it is not necessarily harder to detect uniform split iceberg after introducing the sampling component. In uniform split, the iceberg has larger chance to be captured by the sampling component with poorer estimation, even though, it can still be estimated iceberg.

5.8 Processing Cost

In this section we briefly present the number of operations per packet. Although sketches are different, we can assume they all use CAM (content addressable memory) or hardware hash table whenever they need to query the locally collected elephants for an identity; such process can be done in $O(1)$ time. For count-min and count-sketch, it is too costly to use a heap in SRAM (the number of operations per packet can be as high as 200 using our parameters). The most costly part is the procedure of finding an item already captured and updating its value. In sticky sampling, this number is about 8. Multistage and sample-and-hold can be sustained by SRAM [7]. We find that the number of operations per packet is around 30 for lossy counting, which is also unpromising. In lossy counting algorithm, every new item will be inserted into the sketch, and be deleted if it is not estimated to be an elephant. It is costly to process every packet in such a costly manner.

6 Discussion

Our combination of local sketches and sampling out-performs all the naive approaches for detecting global icebergs. The naive sampling approach is accurate when the sampling interval is small. However, the prohibitive communication cost precludes such a solution. The naive iceberg approach misses the distributed global icebergs that appear as non-elephants at some monitors.

We compared different sketches as well as different combinations of sampling and sketching. We conclude that SSS using QC with a large sampling interval is the best choice. Not only does this solution give a small probability of false alarm and average relative error, but also requires only a small number of operations per packet. Its communication cost is only 1% of the naive approach. This does not mean that SSS or QC are always best for all metrics. For instance, count-min often has a smaller false negative probability, with much higher estimation error and probability of false alarm. Also, RS is generally better than QC when uniform sampling interval is small.

We attain some interesting insights from the experiments on the combinations, which help understand global iceberg detection in distributed streams.

First, it is better to follow the design of flow memory in multistage to store local elephants, instead of using heaps, as in count-min or count-sketch. Since multistage can query whether an item is already contained in the sketch, it has the opportunity to directly update the count values of existing icebergs, which improves accuracy.

Second, comparing count-min and count-sketch reveals which method is better for estimating the count values. Although the estimate from count-sketch is unbiased [8], its overall performance is worse than count-min or multistage.

Thirdly, we presented a particular example in which global iceberg detection differs from local elephant detection in Sect. 5.6 . The slight modification of multistage can introduce underestimation, thus leading to increased probability of false negatives in detecting local elephants. However, both the overall detection accuracy and the accuracy for the small icebergs are improved for the global iceberg detection problem.

Finally, although uniformly split icebergs are difficult to detect by any naive iceberg approach, thanks to the sampling component, the QC combination makes uniform split icebergs as detectable as non-uniform split icebergs.

We believe our comparison study provides a first step towards global iceberg detection in distributed streams. The insights we gained will be useful in other experiments with different parameter settings.

Our solution to this problem is not yet optimal. There are some methods we need to explore. For instance, multistage [7] can be made to measure many shorter intervals, which might be more accurate and practical. Also, we did not optimize the process of reporting data sets to the central server, but only focused on combining sampling and sketching in local measurement. Results from [20] might further improve our solution. We will explore these in future work.

7 Conclusion

In this paper we introduce and motivate the study of distributed algorithms for uncovering global icebergs across multiple streams. This is an important problem in the context where resources are limited at both the local collection points as well as the links connecting them to the central aggregator. This is a very useful application for problems such as worm detection, SLA measurements, and DDoS attack containment.

We studied the effect of combining several of the most widely used local heavy-hitter detection algorithms with sampling across the local points to estimate the global icebergs. We performed experiments on both real as well as synthetically constructed data to compare these different sketches and algorithms. Our experiments showed that one combination (QC) of uniform sampling with the multistage algorithm gives the best detection and estimation of global icebergs.

Acknowledgments This work is supported in part by NSF grants CNS-0519745, CNS-0626979, CNS-0716423, and CT-ISG-0716831.

Open Access This article is distributed under the terms of the Creative Commons Attribution Noncommercial License which permits any noncommercial use, distribution, and reproduction in any medium, provided the original author(s) and source are credited.

References

1. Ayres, P.E., Sun, H., Chao, H.J., Lau, W.C.: ALPi: A DDoS defense system for high-speed networks. IEEE J. Sel. Areas Commun. **24**(10), 1864–1876 (2006)
2. Akamai Technologies Inc. (2004). http://www.akamai.com/
3. Cheetancheri, S.G., Agosta, J.M., Dash, D.H., Levitt, K.N., Rowe, J., Schooler, E.M.: A distributed host-based worm detection system. In: Proceedings of the 2006 SIGCOMM Workshop on Large-Scale Attack Defense (LSAD) (2006)
4. Sommers, J., Barford, P., Duffield, N., Ron, A.: Accurate and efficient sla compliance monitoring. In: Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM) (2007)
5. Cisco Netflow (2007). http://www.cisco.com/
6. Cormode, G., Muthukrishnan, S.: An improved data stream summary: the count-min sketch and its applications. J. Algorithms **55**(1), 58–75 (2005)
7. Estan, C., Varghese, G.: New directions in traffic measurement and accounting. In: Proceedings of the ACM SIGCOMM (2002)
8. Charikar, M., Chen, K., Farach-Colton, M.: Finding frequent items in data streams. Theor. Comput. Sci. **312**(1), 3–15 (2004)
9. Arasu, A., Manku, G.S.: Approximate counts and quantiles over sliding windows. In: Proceedings of the ACM Symposium on Principles of Database Systems (PODS) (2004)
10. Raspall, F., Sallent, S., Yufera, J.: Shared-state sampling. In: IMC '06: Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement, pp. 1–14. ACM, New York, NY (2006)
11. Karp, R.M., Shenker, S., Papadimitriou, C.H.: A simple algorithm for finding frequent elements in streams and bags. ACM Trans. Database Syst. **28**(1), 51–55 (2003)
12. Huang, G., Lall, A., Chuah, C.-N., Xu, J.: Uncovering global icebergs in distributed monitors. In: Proceedings of the IEEE IWQoS (2009)
13. Babcock, B., Babu, S., Datar, M., Motwani, R., Thomas, D.: Operator scheduling in data stream systems. VLDB J. **13**(4), 333–353 (2004)
14. Feigenbaum, J., Kannan, S., Strauss, M., Viswanathan, M.: Streaming algorithms for distributed, massive data sets. In: Proceedings of the IEEE FOCS (1999)
15. Cherniack, M., Balakrishnan, H., Balazinska, M., Carney, D., Cetintemel, U., Xing, Y., Zdonik, S.: Scalable distributed stream processing. In: Proceedings of the 2003 CIDR Conference (2003)
16. Cormode, G., Muthukrishnan, S., Yi, K.: Algorithms for distributed functional monitoring. In: Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA) (2008)
17. Manjhi, A., Shkapenyuk, V., Dhamdhere, K., Olston, C.: Finding (recently) frequent items in distributed data streams. In: Proceedings of the IEEE ICDE (2005)
18. Babcock, B., Olston, C.: Distributed top-k monitoring. In: Proceedings of the ACM International Conference on Management of Data (SIGMOD) (2003)
19. Olston, C., Jiang, J., Widom, Jennifer: Adaptive filters for continuous queries over distributed data streams. In: Proceedings of the ACM International Conference on Management of Data (SIGMOD) (2003)
20. Zhao, Q., Ogihara, M., Wang, H., Xu, J.: Finding global icebergs over distributed data sets. In: Proceedings of the ACM Symposium on Principles of Database Systems (PODS) (2006)
21. Cohen, E., Duffield, N., Kaplan, H., Lund, C., Thorup, M.: Sketching unaggregated data streams for subpopulation-size queries. In: PODS '07: Proceedings of the Twenty-Sixth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pp. 253–262. ACM, New York, NY (2007)
22. Cormode, G., Hadjieleftheriou, M.: Finding frequent items in data streams. PVLDB **1**(2), 1530–1541 (2008)
23. Kumar, A., Xu, J.: Sketch guided sampling—using on-line estimates of flow size for adaptive data collection. In: Proceedings of the IEEE INFOCOM (2006)

24. Huang, Guanyao, Lall, Ashwin, Chuah, ChenNee, Xu, Jun: Uncovering global icebergs in distributed streams. Technical Report ECE-CE-2009-1, UCDavis (2009)
25. Internet2 Abilene Network (2005). http://abilene.internet2.edu/
26. Loiseau, P., Gonçalves, P., Girard, S., Forbes, F., Primet, P.V.-B.: Maximum likelihood estimation of the flow size distribution tail index from sampled packet data. In: SIGMETRICS '09: Proceedings of the Eleventh International Joint Conference on Measurement and Modeling of Computer Systems, pp. 263–274 (2009)

Author Biographies

Guanyao Huang is a third year Ph.D. candidate in Department of Electrical Computer Engineering, University of California, Davis. His current research focuses on network measurement and anomaly detection. He completed his undergraduate and postgraduate degree from the University of Science and Technology, China.

Dr. Ashwin Lall is an Assistant Professor in the Department of Mathematics and Computer Science at Denison University. He did postdoctoral work at Georgia Tech, received M.Sc. and Ph.D. degrees from the University of Rochester, and a B.A. from Colgate University. His research interests are in algorithmics, specifically streaming algorithms.

Chen-Nee Chuah is a Professor in Electrical and Computer Engineering at the University of California, Davis. She received her B.S. from Rutgers University, and her M.S. and Ph.D. in Electrical Engineering and Computer Sciences from the University of California, Berkeley. Her research interests include Internet measurements, network management, anomaly detection, and online social networks.

Dr. Jun Xu is an Associate Professor in the College of Computing at Georgia Institute of Technology. He received his Ph.D. in Computer and Information Science from The Ohio State University in 2000. He received the NSF CAREER award in 2003 for his efforts in establishing fundamental lower bound and tradeoff results in networking, and is a co-recipient of the **Best Student Paper Award** from 2004 ACM Sigmetrics/IFIP Performance joint conference.