LD-Sketch: A Distributed Sketching Design for Accurate and Scalable Anomaly Detection in Network Data Streams

Qun Huang and Patrick P. C. Lee
The Chinese University of Hong Kong
{qhuang,pclee}@cse.cuhk.edu.hk

Abstract—**Real-time characterization of traffic anomalies, such as heavy hitters and heavy changers, is critical for the robustness of operational networks, but its accuracy and scalability are challenged by the ever-increasing volume and diversity of network traffic. We address this problem by leveraging parallelization. We propose LD-Sketch, a data structure designed for accurate and scalable traffic anomaly detection using distributed architectures. LD-Sketch combines the classical counter-based and sketch-based techniques, and performs detection in two phases: local detection, which guarantees zero false negatives, and distributed detection, which reduces false positives by aggregating multiple detection results. We derive the error bounds and the space and time complexity for LD-Sketch. We compare LD-Sketch with state-of-the-art sketch-based techniques by conducting experiments on traffic traces from a real-life 3G cellular data network. Our results demonstrate the accuracy and scalability of LD-Sketch over prior approaches.**

I. Introduction

Characterizing traffic anomalies is important for administrators to maintain the robustness of a network. There are two types of anomalies that are of particular interest: flows with persistently large data volume (known as "heavy hitters") and flows with abrupt changes of data volume (known as "heavy changers"), since they may imply the existence of denial-of-service attacks, component failures, or service level agreement violation. Traffic anomalies are detrimental to network robustness and must be detected and suppressed in real-time.

However, today's IP networks continuously grow in size and complexity. Thus, characterizing traffic anomalies in real-time becomes more challenging, particularly in two aspects:

- *Enormous key space.* The complexity of anomaly detection is overwhelmed by the number of keys being monitored. For example, 5-tuple network flows are defined by 104-bit keys (i.e., source/destination IP addresses, source/destination ports, and protocols). Keeping track of 2^{104} flow keys can imply huge memory usage.
- *Line-rate packet processing.* Packet processing must keep pace with the increasing line rate to meet the real-time requirement. Conventional single-processor platforms no longer provide enough computational power to achieve this goal. To improve scalability, anomaly detection needs to be performed on *distributed* packet streams in parallel. However, providing both accuracy and scalability guarantees becomes challenging when aggregating the detection results from multiple sources.

978-1-4799-3360-0/14/$31.00 © 2014 IEEE

Anomaly detection has been extensively studied in the context of data streaming. To deal with the enormous key space, counter-based techniques (e.g., [10], [16]–[18]) and sketch-based techniques (e.g., [2], [3], [5]–[9], [11], [13], [14], [20]) propose space-efficient data structures for anomaly detection and derive error bounds. The counter-based technique uses an associative array to monitor frequent items and is designed for heavy hitter detection; the sketch-based technique projects data items into a subset of buckets in a summary called *sketch* and is designed for both heavy hitter and heavy changer detection. Although theoretically sound, such techniques are mainly studied and evaluated in the single-processor paradigm. With the emergence of distributed streaming architectures (e.g., Flume [1], S4 [19], and Storm [21]), an open issue is to seamlessly parallelize such techniques to achieve accurate and scalable anomaly detection.

In this paper, we study the traffic anomaly detection problem from both theoretical and implementation perspectives. We propose *LD-Sketch*, a novel sketching design that combines the counter-based and sketch-based techniques to accurately and scalably detect heavy hitters and heavy changers using distributed architectures. Its main idea is to augment a sketch in which each bucket uses the counter-based technique to keep track of anomaly candidates in an associative array. We modify the counter-based technique to allow the associative array to be dynamically expandable based on the current number of anomaly candidates associated with the bucket. LD-Sketch performs detection in two phases: (i) *local detection*, which guarantees no false negatives and identify the anomaly candidates (including true anomalies and false positives) in a single compute node called *worker*, and (ii) *distributed detection*, which reduces false positives (with a slight increase in the false negative rate) by combining detection results from multiple workers. Thus, not only do we exploit streaming architectures to improve scalability, but we also use their distributed nature to improve the detection accuracy.

In summary, we make the following contributions:

- We design LD-Sketch, which enables accurate and scalable detection of heavy hitters and heavy changers and is seamlessly deployable in distributed architectures. We derive the error bounds, space complexity, and time complexity when LD-Sketch is used in both local detection (i.e., using a single worker) and distributed detection (i.e., using multiple workers).
- We implement and compare LD-Sketch with state-of-the-

TABLE I
MAJOR NOTATION USED IN THE PAPER.

Notation	Meaning
	Defined in Section II
p	number of remote sites
q	number of workers
$[n]$	key domain
(x, v_x)	data item with key x and value v_x
$S(x)$	sum for key x in an epoch
$D(x)$	difference for key x between two adjacent epochs
U	total sum of all keys in an epoch
ϕ	heavy key threshold
H	maximum number of heavy keys
r	number of rows in a sketch
w	number of buckets in one row in a sketch
(i, j)	the j-th bucket in row i, where $1 \leq i \leq r$ and $1 \leq j \leq w$
f_i	hash function $\{0, 1, \ldots, n-1\} \rightarrow \{1, 2, \cdots, w\}$ for row i
	Defined in Section III
$V_{i,j}$	counter value of bucket (i, j) in the sketch
$A_{i,j}$	associative array in bucket (i, j) ($A_{i,j}[x]$ denotes the counter value of key x)
$l_{i,j}$	maximum length of array $A_{i,j}$
$e_{i,j}$	maximum error for keys in bucket (i, j)
T	expansion parameter
k	current expansion round (starting from zero)
$S^{low}_{i,j}(x)$	lower estimated sum for key x in bucket (i, j)
$S^{up}_{i,j}(x)$	upper estimated sum for key x in bucket (i, j)
$D_{i,j}(x)$	estimated difference for key x in bucket (i, j)
ϵ	approximation parameter in local detection
δ	upper bound of error probability
	Defined in Section IV
γ	approximation parameter in distributed detection
d	number of workers to which a key is distributed

art sketch-based techniques by conducting trace-driven experiments using traces from a real-life 3G cellular network. We show that LD-Sketch achieves higher accuracy than other approaches, and improves accuracy and scalability using multiple workers in a distributed setting.

The rest of the paper proceeds as follows. Section II formulates the heavy hitter/changer detection problem. Sections III and IV describe and analyze the local and distributed detection procedures of LD-Sketch, respectively. Section V presents our trace-driven evaluation results. Section VI reviews related work, and finally Section VII concludes the paper.

II. Problem Formulation

We formulate the problem of heavy hitter/changer detection (which we collectively call *heavy key detection*). Table I summarizes the major notation in this paper.

A. Heavy Key Detection in a Distributed Architecture

We first describe the distributed architecture, as shown in Figure 1, on which our heavy key detection problem is formulated. Our architecture is based on that of [4]. Specifically, we consider an architecture with $p \geq 1$ *remote sites* and $q \geq 1$ *workers*. A remote site is the source of a data stream, while a worker performs heavy key detection based on the streams from multiple remote sites. The architecture has a bipartite structure, in which each remote site can connect to all workers, while there is no communication among the remote sites and among the workers.

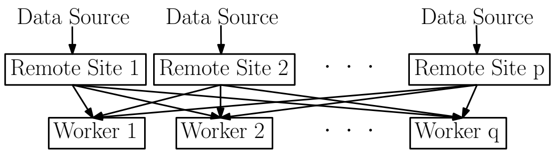


Fig. 1. Distributed architecture for heavy key detection.

We periodically perform heavy key detection on the ordered streams of data items from the p remote sites, and we refer to each time period of detection as an *epoch*. Each data item is represented by a tuple (x, v_x), where x is the key drawn from a domain $[n] = \{0, 1, \cdots, n-1\}$ of size n and v_x is a value associated with x. For example, in network traffic monitoring, x may refer to a 5-tuple flow, and v_x may refer to the byte counts of the flow. In each epoch, let $S(x)$ be the sum of values of key x, and $D(x)$ be the absolute difference of $S(x)$ of key x in the current epoch and in the last epoch. Also, let $U = \sum_{x \in [n]} S(x)$ be the total sum of all keys in the epoch. Our primary goal is to detect the heavy keys whose sums or differences exceed an absolute-value threshold ϕ in an epoch. Specifically, a *heavy hitter* is a key x whose $S(x) \geq \phi$, and a *heavy changer* is a key x whose difference $D(x) \geq \phi$. For ease of presentation, we use the same threshold ϕ for both types of heavy keys, although it can be defined differently.

Let H be the maximum number of heavy keys. Thus, for heavy hitter detection, $H = \frac{U}{\phi}$; for heavy changer detection, $H = \frac{2U}{\phi}$ (i.e., twice the maximum number of heavy hitters in each epoch). Our space and time complexity results are expressed in terms of H, as in prior studies in the literature.

A heavy key detection algorithm typically consists of two procedures: the *update* procedure, which includes the value of each arriving data item into a data structure, and the *detection* procedure, which examines the data structure at the end of each epoch and reports the heavy keys. A good heavy key detection algorithm should introduce (i) small fractions of false positives (i.e., non-heavy keys being treated as heavy keys) and false negatives (i.e., true heavy keys that are not reported), (ii) low space usage of the data structure, and (iii) low time complexity of both update and detection procedures.

B. Counter-based and Sketch-based Techniques

We describe two main classes of techniques that identify the heavy keys using space-efficient data structures: the counter-based and sketch-based techniques.

1) Counter-based Technique: The counter-based technique aims to keep individual counters for a subset of keys. For some performance parameter l, an associative array A with at most $l-1$ key-value counters is defined. Suppose that we have a stream of data items $\{(x, v_x)\}$ and $v_x = 1$ for all x. If x is in A, its counter value $A[x]$ is incremented by one; otherwise, if there are empty counters, a new counter is allocated for x with value initialized to one; if no counter is available, each of other existing counters has its value decremented by one. We remove a key from A if its counter value is zero. The value of each counter remaining in A approximates the true sum of the corresponding key. Lemma 1 shows the error bound of the estimated value [18].

Lemma 1. *Consider a stream of data items $\{(x, v_x)\}$ and $v_x = 1$. In the counter-based technique, if x is kept in A, then $A[x] \leq S(x) \leq A[x] + \frac{U}{l}$; if x is not in A, its estimated value is set to zero and $0 \leq S(x) \leq \frac{U}{l}$.*

The counter-based technique can identify all heavy hitters (without false negatives) with threshold exceeding $\phi = \frac{U}{l}$ by checking if a key has a positive counter value in the associative array. False positives may exist if some non-heavy hitters remain in the array and are not removed by the end of the epoch. To the best of our knowledge, it remains an open issue how the counter-based technique is applied to heavy changer detection, mainly because the associative array is not linear and we cannot combine two arrays of two epochs to describe the differences of data items.

2) Sketch-based Techniques: Sketch-based techniques process a data stream in sub-linear space, and can be used to identify both heavy hitters and heavy changers. A sketch is a small summary data structure projected from a large set. Specifically, we consider a sketch with r rows. Each row i $(1 \leq i \leq r)$ is associated with w buckets and an independent 2-universal hash function f_i that hashes a key to one of the w buckets. We denote the j-th bucket $(1 \leq j \leq w)$ in row i by (i, j). Each bucket (i, j) is associated with a counter value $V_{i,j}$ initialized at zero. For each data item (x, v_x), the update procedure hashes key x to one of the buckets in each of the r rows, and increments the counter value by v_x. By Markov's inequality, we have the following lemma:

Lemma 2. *In sketch-based techniques,* $\Pr\{V_{i,j} \geq \nu\} \leq \frac{U}{w\nu}$ *for any sufficient large $\nu > \frac{U}{w}$.*

The sketch-based technique can identify all heavy hitters (without false negatives) with threshold exceeding ϕ by checking if the corresponding buckets of all r rows have values exceeding ϕ. False positives may exist if non-heavy hitters are hashed to the buckets with values above the threshold in all rows. For heavy changer detection, we compute the differences of counter values of each bucket in the two sketches in two adjacent epochs. By the linear property of a sketch, the difference of each bucket is also the sum of the differences of the keys hashed to the bucket. If a key has differences in the corresponding buckets of all r rows (or a subset of them [5]) exceeding ϕ, it is reported as a heavy changer. False negatives may also exist in heavy changer detection, for example, when a heavy changer with a positive change greater than ϕ and another heavy changer with a negative change less than $-\phi$ are hashed to the same bucket.

C. Our Approach: Overview

This work proposes LD-Sketch, a design that leverages the distributed architecture to achieve accurate and scalable heavy key detection. It follows a sketching design that combines the classical counter-based and sketch-based techniques. It builds on two phases: local detection and distributed detection, which we describe in Sections III and IV, respectively.

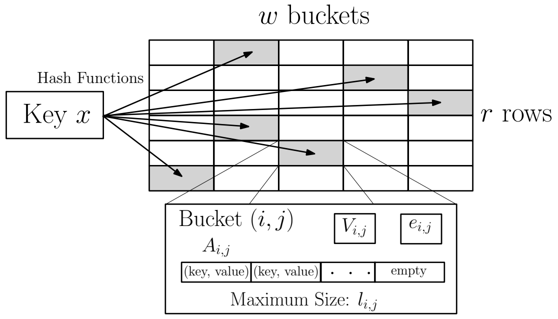


Fig. 2. Structure of LD-Sketch.

III. Local Detection

We present the local detection approach of LD-Sketch that aims to identify the heavy hitters and heavy changers in a single worker. We also analyze the error bounds, space complexity, and time complexity of the local detection approach.

A. Data Structure

We first describe the data structure of LD-Sketch, as shown in Figure 2. An LD-Sketch is composed of r rows with w buckets each. Each bucket (i, j) (where $1 \leq i \leq r$ and $1 \leq j \leq w$) corresponds to the j-th bucket in row i, and is associated with a counter value $V_{i,j}$, which will be incremented by v_x for every incoming key x. We augment each bucket (i, j) with three additional components, including: (i) $A_{i,j}$, which denotes the associative array used in counter-based detection, (ii) $l_{i,j}$, which denotes the maximum length of $A_{i,j}$, and (iii) $e_{i,j}$, which denotes the maximum estimation error for the true sums of the keys hashed to bucket (i, j).

Our main idea is to use $A_{i,j}$ to keep track of the heavy key candidates that are hashed to the bucket (i, j). Thus, when we restore all heavy keys in the detection procedure, we only inspect the tracked heavy key candidates in all $A_{i,j}$'s, and this significantly improves the accuracy and performance of our heavy key detection. One new extension to the counter-based technique of [18] is that we dynamically increase $l_{i,j}$ (i.e., the maximum size of $A_{i,j}$) as $V_{i,j}$ increases. We call this approach *dynamical expansion*. We define an expansion parameter T as a function of the threshold ϕ (see Section III-B), and the current expansion round $k = \lfloor V_{i,j}/T \rfloor \geq 0$, such that when k is incremented, we increase $l_{i,j}$. We assume that $T > v_x$ for any data item x, so k is incremented by at most one for each new data item x. The dynamic expansion approach trades memory for accuracy, i.e., by keeping track of more high-valued keys, we can restore the heavy keys more accurately.

Algorithm 1 details how we update a data item to an LD-Sketch. Initially, $V_{i,j}$, $l_{i,j}$, and $e_{i,j}$ are set to zero and $A_{i,j}$ is empty for each bucket (i, j) (where $1 \leq i \leq r$ and $1 \leq j \leq w$). For a data item (x, v_x), we hash the key to some bucket (i, j) via a hash function f_i for each row i, and

Algorithm 1 LD-Sketch Update Algorithm

```
Input: data item $(x, v_x)$; expansion parameter $T$
1: function UPDATEBUCKET($x, v_x, i, j$)
2:     $V_{i,j} = V_{i,j} + v_x$
3:     if $x \in A_{i,j}$ then
4:         $A_{i,j}[x] = A_{i,j}[x] + v_x$
5:     else if $A_{i,j}$ has less than $l_{i,j}$ counters then
6:         Insert $x$ to $A_{i,j}$ and set $A_{i,j}[x] = v_x$
7:     else
8:         $k = \lfloor V_{i,j}/T \rfloor$
9:         if $(k+1)(k+2) - 1 \leq l_{i,j}$ then
10:            $\hat{e} = \min(v_x, \min_{y \in A_{i,j}} A_{i,j}[y])$
11:            $e_{i,j} = e_{i,j} + \hat{e}$
12:            for all key $y \in A_{i,j}$ do
13:                $A_{i,j}[y] = A_{i,j}[y] - \hat{e}$
14:                if $A_{i,j}[y] \leq 0$ then
15:                    Remove $y$ from $A_{i,j}$
16:                end if
17:            end for
18:            if $v_x > \hat{e}$ then
19:                Insert $x$ to $A_{i,j}$ and set $L_{i,j}(x) = v_x - \hat{e}$
20:            end if
21:        else                                   ▷ $l_{i,j} < (k+1)(k+2) - 1$
22:            add $(k+1)(k+2) - 1 - l_{i,j}$ new counters to $A_{i,j}$
23:            $l_{i,j} = (k+1)(k+2) - 1$
24:            Insert $x$ to $A_{i,j}$ and set $A_{i,j}[x] = v_x$
25:        end if
26:    end if
27: end function
28:
29: procedure UPDATE
30:    for data item $(x, v_x)$ do
31:        for row $i = 1, 2, \ldots, r$ do
32:            $j = f_i(x)$
33:            UPDATEBUCKET($x, v_x, i, j$)
34:        end for
35:    end for
36: end procedure
```

call the function UPDATEBUCKET. In essence, the function UPDATEBUCKET builds on the counter-based technique, with an extension of enabling dynamic expansion. Specifically, let us consider bucket (i, j). If key x is in $A_{i,j}$, we increment the counter $A_{i,j}[x]$ (Line 4); or if $A_{i,j}$ is not yet full, we insert x to $A_{i,j}$ (Line 6). Otherwise, if $A_{i,j}$ is full, we either decrement the keys in $A_{i,j}$ or expand $A_{i,j}$. We consider both cases below.

We first describe how we decrement the keys (Lines 10-20), and the steps are similar to the original counter-based technique [18]. The decrement value $\hat{e}$ is chosen to be the minimum of v_x and the minimum value of $A_{i,j}$ (Line 10). We first add $\hat{e}$ to $e_{i,j}$ (which will be used in the detection procedure) (Line 11). Then we decrement all keys in $A_{i,j}$ by $\hat{e}$ and remove the keys whose values are no more than zero (Lines 12-17). If v_x is not completely decremented, the residue $v_x - \hat{e}$ can be inserted to $A_{i,j}$, given that the some key must have been removed (Lines 18-20).

We next describe the dynamic expansion of $A_{i,j}$ (Lines 22-24). We keep track of the current expansion round $k = \lfloor V_{i,j}/T \rfloor$ (Line 8). We set $l_{i,j} = (k+1)(k+2) - 1$. When $A_{i,j}$ is full, if $l_{i,j} < (k+1)(k+2) - 1$, which happens when k is incremented (by at most one as stated above), then we add new counters to $A_{i,j}$.

We use LD-Sketch to estimate the true sum $S(x)$ of each key x. Here, LD-Sketch produces a pair of estimates for each key x and each bucket (i, j): the lower estimate $S_{i,j}^{low}(x)$ and the upper estimate $S_{i,j}^{up}(x)$. If x is in $A_{i,j}$, we set $S_{i,j}^{low}(x) = A_{i,j}[x]$; otherwise $S_{i,j}^{low}(x) = 0$. Also, we set $S_{i,j}^{up}(x) = S_{i,j}^{low}(x) + e_{i,j}$.

B. Heavy Key Detection

We now explain how we identify heavy hitters and heavy changers using LD-Sketch.

For heavy hitter detection, we use a single LD-Sketch with expansion parameter $T = \phi$. At the end of each epoch, we examine every bucket (i, j) in the sketch. We identify every bucket (i, j) with $V_{i,j} \geq \phi$, and examine the keys kept in $A_{i,j}$. A key x is reported as a heavy hitter if $S_{i,j}^{up}(x) \geq \phi$ for all row i, where $1 \leq i \leq r$, and $j = f_i(x)$.

For heavy changer detection, we maintain two LD-Sketches for two adjacent epochs. Both sketches set the expansion parameter $T = \epsilon\phi$, where $\epsilon \in (0, 1]$ denotes the approximation parameter that bounds the error rates. At the end of the second epoch, we identify every bucket (i, j) with $V_{i,j} \geq \phi$ in at least one epoch, and examine the keys of $A_{i,j}$ in both sketches. We obtain the lower and upper estimates in the first epoch and those in the second epoch, denoted by $S_{i,j}^{low,1}(x)$, $S_{i,j}^{up,1}(x)$, $S_{i,j}^{low,2}(x)$, and $S_{i,j}^{up,2}(x)$, respectively. The estimated change is given by $D_{i,j}(x) = \max\{S_{i,j}^{up,1}(x) - S_{i,j}^{low,2}(x), S_{i,j}^{up,2}(x) - S_{i,j}^{low,1}(x)\}$. A key x is reported as a heavy changer if $D_{i,j}(x) \geq \phi$ for all row i, where $1 \leq i \leq r$, and $j = f_i(x)$.

C. Analysis

In this section, we present a theoretical analysis of the local detection of LD-Sketch as described in Section III-B. We analyze the error rates, space complexity, and time complexity.

First, Lemmas 3 and 4 describe the range of the estimated sum of a key.

Lemma 3. *$S_{i,j}^{low}(x) \leq S(x) \leq S_{i,j}^{up}(x)$ for every key x and bucket (i, j).*

Proof: From Algorithm 1, $S(x) \geq A_{i,j}[x] = S_{i,j}^{low}(x)$ since $A_{i,j}[x]$ is never incremented due to other items not belonging to x. Also, $A_{i,j}[x]$ is decremented by at most $e_{i,j}$, so $A_{i,j}[x] \geq S(x) - e_{i,j}$ and hence $S(x) \leq S_{i,j}^{up}(x)$. ■

Lemma 4. *For bucket (i, j), if $kT \leq V_{i,j} < (k+1)T$,*

$$S(x) \leq S_{i,j}^{low}(x) + (\frac{k+1}{k+2})T.$$

$$S_{i,j}^{up}(x) \leq (\frac{k+1}{k+2})T + (1 - \frac{1}{(k+1)(k+2)})S(x).$$

Proof: By Lemma 3, $S(x) \leq S_{i,j}^{low}(x) + e_{i,j}$. In each expansion round κ ($0 \leq \kappa \leq k$), $A_{i,j}$ contains $l_{i,j} = (\kappa + 1)(\kappa + 2) - 1$ counters, so $e_{i,j}$ is incremented by at most $\frac{T}{l_{i,j}+1} = \frac{T}{(\kappa+1)(\kappa+2)}$ (by Lemma 1 [18]). Thus, before the expansion round $k+1$, $e_{i,j}$ is at most $\sum_{\kappa=0}^{k} \frac{T}{(\kappa+1)(\kappa+2)} = \sum_{\kappa=0}^{k}(\frac{1}{\kappa+1} - \frac{1}{\kappa+2})T = (\frac{k+1}{k+2})T$.

For $S_{i,j}^{up}(x)$, let $e'_{i,j} = S(x) - S_{i,j}^{low}(x)$ be the decrements of $A_{i,j}[x]$. Thus, $e_{i,j} - e'_{i,j}$ corresponds to the decrements of $A_{i,j}[y]$ for any $y \neq x$ when $A_{i,j}[x]$ is not decremented, and is contributed by the values $V_{i,j} - S(x)$. Note that $e_{i,j} - e'_{i,j}$ achieves maximum if $V_{i,j} - S(x) > kT$, and its value is at most $\sum_{\kappa=0}^{k-1} \frac{T}{(\kappa+1)(\kappa+2)} + \frac{V_{i,j}-S(x)-kT}{(k+1)(k+2)} \leq (\frac{k}{k+1})T + \frac{T-S(x)}{(k+1)(k+2)} = (\frac{k+1}{k+2})T - \frac{S(x)}{(k+1)(k+2)}$. Since $S_{i,j}^{up}(x) = S_{i,j}^{low}(x) + e_{i,j} = S(x) + (e_{i,j} - e'_{i,j})$. The results follow. ■

Based on the range of the estimated sum of a key, Lemma 5 argues that our local detection has zero false negatives.

Lemma 5. *Using LD-Sketch, if key x has $S(x) \geq \phi$, it must be reported as a heavy hitter; if x has $D(x) \geq \phi$, it must be reported as a heavy changer.*

Proof: By Lemma 4, if $S(x) \geq \phi$, then for any bucket (i,j) associated with x, we must have $A_{i,j}[x] = S_{i,j}^{low}(x) > S(x) - T \geq 0$ (note that $T = \phi$ for heavy hitter detection and $T = \epsilon\phi$ for heavy changer detection). For heavy changer detection, if $D(x) \geq \phi$, there must be at least one epoch where $S(x) \geq 0$. Therefore, x must be kept in the associative array of its corresponding buckets.

By Lemma 3, we know $S(x) \leq S_{i,j}^{up}$ for every bucket (i,j). If $S(x) \geq \phi$, then $S_{i,j}^{up}(x) \geq \phi$, so x must be reported as a heavy hitter. Also, $D(x) \leq D_{i,j}(x)$ for every bucket (i,j). If $D(x) \geq \phi$, then $D_{i,j}(x) \geq \phi$, so x must be reported as a heavy changer. ■

Lemmas 6 and 7 discuss the false positive rates of heavy hitter detection and heavy changer detection, respectively.

Lemma 6. *For key x with $S(x) < \phi$, it is reported as a heavy hitter with probability at most $(\frac{U}{w\phi})^r$.*

Proof: In heavy hitter detection, we set $T = \phi$. For key x with $S(x) < \phi$, there always exists an integer $k \geq 0$ such that $S(x) < \frac{\phi}{k+2-\frac{1}{k+1}}$. If $V_{i,j} < (k+1)\phi$, by Lemma 4, $S_{i,j}^{up}(x) < (\frac{k+1}{k+2})\phi + (1 - \frac{1}{(k+1)(k+2)})S(x) = (\frac{k+1}{k+2})\phi + (1 - \frac{1}{(k+1)(k+2)})\frac{\phi}{k+2-\frac{1}{k+1}} = \phi$. So x is not reported as a heavy hitter. On the other hand, x is reported as a heavy hitter only if $V_{i,j} \geq (k+1)\phi$ for all row i. By Lemma 2, the probability that $V_{i,j} \geq (k+1)\phi$ is at most $(\frac{U}{w(k+1)\phi})$. Since the r hash functions are independent, the probability that x is reported as a heavy hitter is $(\frac{U}{w(k+1)\phi})^r \leq (\frac{U}{w\phi})^r$. ■

Lemma 7. *For key x with $D(x) < (1-\epsilon)\phi$, it is reported as a heavy changer with probability at most $(\frac{2U}{w\epsilon\phi})^r$.*

Proof: For heavy changer detection, we set $T = \epsilon\phi$. For key x with $D(x) < (1-\epsilon)\phi$, there always exits an integer $k \geq 0$ such that $D(x) < (1 - 2\epsilon(\frac{k+1}{k+2}))\phi$. If $V_{i,j} < (k+1)\epsilon\phi$ in two sketches, by Lemma 4, the error of $S(x)$ in each epoch is at most $\frac{k+1}{k+2}T$. So the estimated change is less than $(1 - 2(\frac{k+1}{k+2})\epsilon)\phi + 2\frac{k+1}{k+2}\epsilon\phi \leq \phi$, implying that x is not reported as a heavy changer.

Key x is reported as a heavy changer only if $V_{i,j} \geq (k+1)\epsilon\phi$ in at least one sketch. The probability of $V_{i,j} \geq (k+1)\epsilon\phi$ in at least one sketch is at most $\frac{2U}{w(k+1)\epsilon\phi}$. Since the r hash functions are independent, the probability that x is reported as a heavy changer is at most $(\frac{2U}{w(k+1)\epsilon\phi})^r \leq (\frac{2U}{w\epsilon\phi})^r$. ■

We now discuss the space complexity of LD-Sketch. Lemma 8 shows the worst-cast expectation of $l_{i,j}$.

Lemma 8. *The expected value of $l_{i,j}$ is given by $E[l_{i,j}] = O(\frac{U^2}{w^2T^2})$.*

Proof: In the worst case, the associative array $A_{i,j}$ is expanded as large as possible, so $l_{i,j} = O(k^2) = O((\frac{V_{i,j}}{T})^2)$. Then we consider $E[V_{i,j}^2]$, where $E[V_{i,j}^2] = E[V_{i,j}]^2 + Var[V_{i,j}]$.

Let Y_x be an indicator variable such that $Y_x = 1$ if key x is hashed to bucket (i,j), or $Y_x = 0$ otherwise. Thus, $V_{i,j} = \sum_{x\in[n]} S(x)Y_x$. Given that the keys are uniformly hashed to the buckets, we have $E[Y_x] = 1/w$ and $Var[Y_x] = \frac{w-1}{w^2}$. Thus, $E[V_{i,j}] = \frac{U}{w}$.

Also, since all keys are hashed independently, $Var[V_{i,j}] = Var[\sum_{x\in[n]} S(x)Y_x] = \frac{w-1}{w^2}\sum_{x\in[n]}(S(x))^2$. Since w is much smaller than the number of available keys n, we have $w\sum_{x\in[n]}(S(x))^2 < (\sum_{x\in[n]} S(x))^2 = U^2$. Hence $Var[V_{i,j}] = O(\frac{w-1}{w^3}U^2)$.

Thus, combining the two terms, $E[l_{i,j}] = O(\frac{U^2}{w^2T^2})$. ■

By Lemma 8, the expected space of an LD-Sketch is the total size of all buckets of the sketch and all associative arrays, and is given by $O(rw(1 + l_{i,j})) = O(r(w + \frac{U^2}{wT^2}))$.

Finally, we examine the time complexity of both update and detection procedures. To update a data item, we update r buckets of the sketch, and for each bucket (i,j), we perform $l_{i,j}$ operations of the associative array. Thus, the time complexity for updating a data item is $O(r(1 + l_{i,j})) = O(r(1 + \frac{U^2}{w^2T^2}))$. For the detection procedure, we enumerate all buckets in an LD-Sketch and the associative arrays. The time complexity of detection is $O(rw(1 + l_{i,j})) = O(r(w + \frac{U^2}{wT^2}))$.

We now propose the parameter selection of LD-Sketch. As stated above, we set $T = \phi$ and $T = \epsilon\phi$ for heavy hitter and heavy changer detection, respectively. We express our results in terms of H, where $H = \frac{U}{\phi}$ for heavy hitter detection and $H = \frac{2U}{\phi}$ for heavy changer detection (see Section II-A). The selection of r and w takes two parameters ϵ and δ as inputs, similar to the work [8]. For heavy hitter detection, LD-Sketch selects $w = 2H, r = \log\frac{1}{\delta}$. For heavy changer detection, LD-Sketch selects $w = \frac{2H}{\epsilon}$ and $r = \log\frac{1}{\delta}$.

With the above lemmas and the selected parameters w, r and T, Theorems 1 and 2 summarize the error bounds, space complexity, and time complexity of LD-Sketch in heavy hitter and heavy changer detection, respectively.

Theorem 1. *Consider an LD-Sketch with $w = 2H, r = \log\frac{1}{\delta}$ and $T = \phi$. It reports all heavy hitters. A non-heavy hitter is reported with probability at most δ. The expected space is $O(H\log\frac{1}{\delta})$. The expected time complexity of updating a data item is $O(\log\frac{1}{\delta})$, and that of detection is $O(H\log\frac{1}{\delta})$.*

Theorem 2. *Consider two LD-Sketches with $w = \frac{2H}{\epsilon}, r =$*

TABLE II
COMPARISON OF EXISTING WORKS WITH LD-SKETCH.

Method	Space	Update Time	Detect Time
Count-Min [9]	$O(\frac{H}{\epsilon}\log\frac{1}{\delta})$	$O(\log\frac{1}{\delta})$	$O(n)$
Combinatorial Group Testing [8]	$O(\frac{H}{\epsilon}\log n\log\frac{1}{\delta})$	$O(\log n\log\frac{1}{\delta})$	$O(\frac{H}{\epsilon}\log n\log\frac{1}{\delta})$
Reversible Sketch [20]	$O(\frac{\log n^{\Theta(1)}}{\log\log n})$	$O(\log n)$	$O(\frac{H}{\epsilon}n^{\frac{3}{\log\log n}}\log\log n)$
SeqHash [2]	$O(\frac{H}{\epsilon}\log n\log\frac{1}{\delta})$	$O(\log n\log\frac{1}{\delta})$	$O(\frac{H}{\epsilon}\log n\log\frac{1}{\delta})$
Fast Sketch [14]	$O(\frac{H}{\epsilon}\log\frac{1}{\delta}\log\frac{n\epsilon}{H\log\frac{1}{\delta}})$	$O(\log\frac{1}{\delta}\log\frac{n\epsilon}{H\log\frac{1}{\delta}})$	$O(\frac{H}{\epsilon}\log^3\frac{1}{\delta}\log\frac{n\epsilon}{H\log\frac{1}{\delta}})$
LD-Sketch	$O(\frac{H}{\epsilon}\log\frac{1}{\delta})$	$O(\log\frac{1}{\delta})$	$O(\frac{H}{\epsilon}\log\frac{1}{\delta})$

$\log\frac{1}{\delta}$ *and* $T = \epsilon\phi$. *They report all heavy changers. A non-heavy changer with difference less than* $(1-\epsilon)\phi$ *is reported with probability at most* δ. *The expected space complexity is* $O(\frac{H}{\epsilon}\log\frac{1}{\delta})$. *The expected time complexity of updating a data item is* $\log\frac{1}{\delta}$, *and that of detection is* $O(\frac{H}{\epsilon}\log\frac{1}{\delta})$.

D. Comparison with Other Sketch-Based Techniques

We compare LD-Sketch with state-of-the-art techniques. In the interest of space, we only consider heavy changer detection. We focus on the sketch-based techniques, which provide error bounds with two parameters: $0 < \epsilon \le 1$ and $0 < \delta < 1$, such that the keys with true difference outside the range $[(1-\epsilon)\phi, (1+\epsilon)\phi)]$ are detected with error probability at most δ. Note that LD-Sketch guarantees zero false negatives for heavy changer detection.

Table II lists the space and time complexity of heavy changer detection for different approaches, in terms of ϕ, ϵ, δ, and H. Count-Min consumes $O(\frac{H}{\epsilon}\log\frac{1}{\delta})$ of memory but requires $O(n)$ time to recover heavy keys. Combinatorial Group Testing, Reversible Sketch, SeqHash, and Fast Sketch aims to reduce the detection time, but they all have a $\log n$ term in the memory complexity to keep track of the heavy keys inside the sketch structure, so that the heavy keys can be recovered from the sketch structure. On the other hand, LD-Sketch has the same memory complexity as Count-Min by keeping the heavy key candidates in the associative arrays, while its update and detection time complexity is comparable to other approaches.

IV. DISTRIBUTED DETECTION

In this section, we elaborate how we perform heavy key detection in a distributed architecture via LD-Sketch.

A. Design

We design our distributed scheme as follows. We deploy an LD-Sketch in each of the q workers. Then we partition the data stream in each of the p remote sites using a two-step approach. First, for a data item (x, v_x), a remote site hashes key x to a set of d $(1 \le d \le q)$ workers for some design parameter d. This selection is deterministic in the sense that the same set of d workers will always be selected for all data items with the same key x. Second, the remote site uniformly selects one of the d workers and forwards it the data item. The partitioning functions in both steps are identical in all remote sites. Since each of the d workers receives on average $1/d$ of the total value for each key, we set the heavy key threshold to be $(1-\gamma)\frac{\phi}{d}$ for some approximation parameter $0 \le \gamma < 1$. Finally, if a key is reported as a heavy key by all the d workers, then we report the key as a heavy key in our final results.

The design parameter d in our distributed scheme determines the accuracy of heavy key detection. As d increases, the false positive rate decreases until some turning point is reached. We also introduce a small false negative rate, since some of the d workers may receive less than $(1-\gamma)\frac{\phi}{d}$ of the total value if the partition is not perfectly even. We study the impact of d on the accuracy in the next subsection.

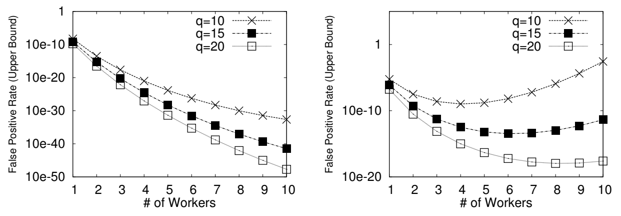
B. Analysis

We conduct theoretical analysis on LD-Sketch in a distributed setting. Our results are derived from those in Section III by choosing new parameters for the distributed scheme.

We first consider the space and time complexity. Suppose that each LD-Sketch has r rows and w buckets. For the total sum U and the threshold ϕ in local detection, we replace them with $\frac{U}{q}$ and $(1-\gamma)\frac{\phi}{d}$, respectively. We assume that the worker selection for each data item is independent. By similar arguments in Lemma 8, the expectation of the term U^2 in the space and time complexity can be replaced with $O(\frac{U^2}{q^2})$, assuming that q is far smaller than the number of available keys n. Using the above arguments, the space complexity is $O(r(w + \frac{U^2}{wT^2q^2}))$. The time complexity of the update procedure is $O(r(1 + \frac{U^2}{w^2T^2q^2}))$, and that of the detection procedure is $O(r(w+\frac{U^2}{wT^2q^2}))$. We set $T = (1-\gamma)\frac{\phi}{d}$ for heavy hitter detection and $T = (1-\gamma)\frac{\epsilon\phi}{d}$ for heavy changer detection. Since $1 \le d \le q$ and γ is typically small, we actually reduce both the space and time complexity of local detection.

For the false positive rate, it can be derived from Lemmas 6 and 7 by replacing U and ϕ with $\frac{U}{q}$ and $(1-\gamma)\frac{\phi}{d}$, respectively. Note that the distributed scheme further reduces the false positive rate since a non-heavy key needs to be reported by all d workers. Thus, the corresponding false positive rate is at most $(\frac{dU}{qw(1-\gamma)\phi})^{rd}$ for heavy hitter detection and at most $(\frac{2dU}{qw(1-\gamma)\epsilon\phi})^{rd}$ for heavy changer detection.

We illustrate how the false positive rate varies with different values of d. As a case study, we pick $\phi = 0.001U$, so there are at most $H = \frac{U}{\phi} = 1000$ heavy hitters and $H = \frac{2U}{\phi} = 2000$ heavy changers (see Section II-A). We fix $\gamma = 0$, $\epsilon = 0.5$, $r = 5$, and $w = 4500$. Figures 3(a) and 3(b) show the upper bounds of the false positive rate for heavy hitter and heavy changer detection, respectively. Note that the upper bound of the false positive rate can be viewed as the function $(cd)^{rd}$, for



(a) Heavy Hitter (b) Heavy Changer

Fig. 3. False positive rate of distributed scheme versus d.

some constant c. As d increases, if $\log(cd) < -1$, the upper bound of the false positive rate decreases; if $\log(cd) > -1$, the upper bound increases exponentially, as shown in Figure 3(b).

For the false negative rate, Lemma 9 shows that it can be bounded in a distributed setting.

Lemma 9. *For a heavy key x, it will be missed at probability at most $1-(1-e^{-\frac{\phi}{2d}\gamma^2})^d$.*

Proof: Let X be the random variable of the sum of values of the heavy key x received by a worker. We first consider heavy hitter detection. Note that $E(X) = \frac{S(x)}{d} \geq \frac{\phi}{d}$. By Chernoff bound, $Pr\{X < (1-\gamma)\frac{\phi}{d}\} < e^{-\frac{E(X)}{2}(1-(1-\gamma)\frac{d\phi}{E(X)})^2} \leq e^{-\frac{\phi}{2d}\gamma^2}$. This is also the probability of missing the key in a single worker. Thus, the probability of missing the key in at least one worker is at most $1-(1-e^{-\frac{\phi}{2d}\gamma^2})^d$.

For heavy changer detection, we denote X as the difference of the heavy key x. Since $E(X) \geq \frac{\phi}{d}$. We can apply the same results as above. ∎

Finally, we propose the parameter selection for the distributed scheme. The selection involves H, ϕ, ϵ, δ, and γ as inputs. For heavy hitter detection, LD-Sketch selects $w = \frac{2dH}{(1-\gamma)q}$, $r = \log\frac{1}{d\delta}$, and $T = \frac{(1-\gamma)\phi}{d}$; for heavy changer detection, it selects $w = \frac{2dH}{(1-\gamma)q\epsilon}$, $r = \log\frac{1}{d\delta}$, and $T = \frac{(1-\gamma)\epsilon\phi}{d}$. With the selected parameters, the following theorems summarize the error bounds, space complexity, and time complexity of the distributed scheme of LD-Sketch.

Theorem 3. *Consider an LD-Sketch with $w = \frac{2dH}{(1-\gamma)q}$, $r = \log\frac{1}{d\delta}$, and $T = \frac{(1-\gamma)\phi}{d}$. A heavy hitter is missed with probability at most $1-(1-e^{-\frac{\phi}{2d}\gamma^2})^d$, while a non-heavy hitter with the true sum less than $(1-\gamma)\phi$ is reported with probability at most δ. The expected space complexity is $O(\frac{dH}{q(1-\gamma)}\log\frac{1}{d\delta})$. The expected time complexity of updating a data item is $O(\log\frac{1}{d\delta})$, and that of detection is $O(\frac{dH}{q(1-\gamma)}\log\frac{1}{d\delta})$.*

Theorem 4. *Consider two LD-Sketches with $w = \frac{2dH}{(1-\gamma)q\epsilon}$, $r = \log\frac{1}{d\delta}$, and $T = \frac{(1-\gamma)\epsilon\phi}{d}$. A heavy changer is missed with probability at most $1-(1-e^{-\frac{\phi}{2d}\gamma^2})^d$, while a non-heavy changer with the true difference less than $(1-\gamma)(1-\epsilon)\phi$ is reported with probability at most δ. The expected space complexity is $O(\frac{dH}{q(1-\gamma)\epsilon}\log\frac{1}{d\delta})$. The expected time complexity of updating a data item is $\log\frac{1}{d\delta}$, and that of detection is $O(\frac{dH}{q(1-\gamma)\epsilon}\log\frac{1}{d\delta})$.*

V. Experiments

In this section, we present results based on trace-driven experiments. Our goal is to demonstrate the accuracy and scalability of LD-Sketch in a distributed setting.

Implementation. We implement a distributed streaming architecture in C++ with remote sites and workers, each of which runs as a software process. To eliminate network I/O overhead in our evaluation, we deploy our architecture in a multi-core testbed, where a remote site and a worker is connected via an in-memory ring buffer. We implement LD-Sketch in each worker.

Testbed and evaluation methodologies. Our testbed is a multi-core server with 12 physical 2.93GHz CPU cores and 50GB RAM. The server runs Linux 2.6.32. We compile our code using GCC 4.6 with the -O3 option.

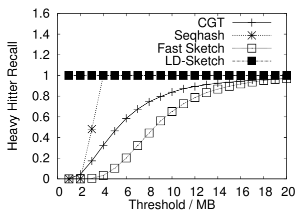
We run our evaluation on real IP packet header traces from a commercial 3G UMTS network in mainland China in December 2010 (the same traces as in prior work [12]). We select the most heavy-loaded six hours of traces for our evaluation. The traces contain 1.1 billion packets that account for a total of around 600GB of traffic. We configure $p = 3$ remote sites and a number of q workers (varied in our experiments). We divide the traces into three sub-traces in a round-robin manner, and each sub-trace is replayed by one of the three remote sites. Each data item corresponds to a packet, where the key is set as the 64-bit source and destination IP address pair and the value is set as the IP payload size. We set the epoch length for heavy key detection as 10 minutes, while we also verify the results for different epoch lengths and make consistent observations. Our results are averaged over all 36 epochs of the six-hour traces.

We compare LD-Sketch with several state-of-the-art sketch-based techniques, including Combinational Group Testing (CGT) [8], SeqHash [2], and Fast Sketch [14] in both heavy hitter and heavy changer detection. We do not consider the counter-based technique, which only supports heavy hitter detection (see Section II-B). Note that all the above sketch-based techniques, except LD-Sketch, are only designed for local detection (i.e., using one worker).

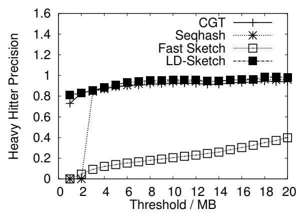
Metrics. We consider the following metrics:

- *Recall:* It is the ratio of the number of true heavy keys returned to the actual number of true heavy keys. A higher recall means a lower false negative rate.
- *Precision:* It is the ratio of the number of true heavy keys returned to the total number of all heavy keys returned (including true heavy keys and false positives). A higher precision means a lower false positive rate.
- *Update throughput:* It is the total IP payload size divided by the total time to update the data items. Before measurements, we load the traces to memory to eliminate disk overhead. We also bind the worker processes to CPU cores to avoid context switching.

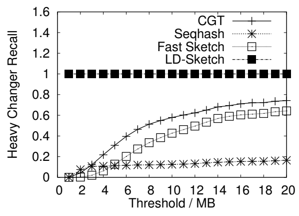
In the interest of space, we do not present the throughput results of the detection procedure. Our experience is that the detection time is significantly less than that of updating all



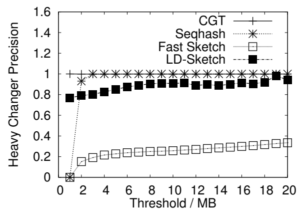
(a) Heavy Hitter, Recall



(b) Heavy Hitter, Precision

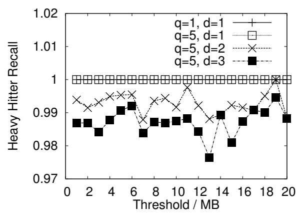


(c) Heavy Changer, Recall

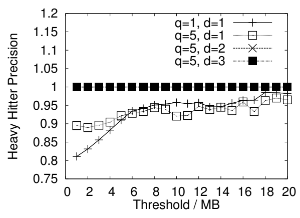


(d) Heavy Changer, Precision

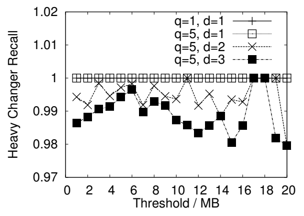
Fig. 4. Experiment 1 (Accuracy for local detection).



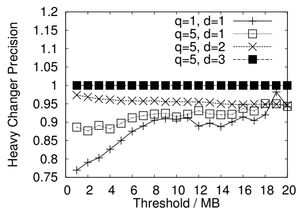
(a) Heavy Hitter, Recall



(b) Heavy Hitter, Precision



(c) Heavy Changer, Recall



(d) Heavy Changer, Precision

Fig. 5. Experiment 2 (Accuracy for distributed detection of LD-Sketch).

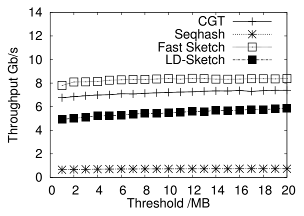
data items in one epoch.

Experiment 1 (Accuracy for local detection). In this experiment, we compare different detection approaches by configuring them with the same amount of memory. We fix $M = 10^6$ counters in the data structure for each approach. For CGT, SeqHash, and Fast Sketch, we fix $\delta = \frac{1}{4}$ and allocate the M counters to the buckets accordingly. For LD-Sketch, we estimate the length of an associated array to be the upper bound $2\frac{U^2}{w^2T^2}$ (see Section III) and set w that satisfies the memory requirement.

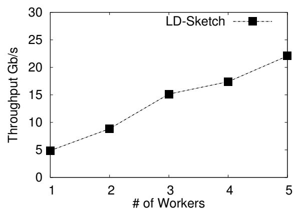
Figure 4 shows the results. We see that all CGT, SeqHash, and Fast Sketch have low recall (see Figures 4(a) and 4(c)), because they need more memory to recover all heavy keys (see Table II). With insufficient memory, they have high false negative rates. Fast Sketch has low precision, since it identifies heavy quotients (i.e., prefixes of heavy keys) and may return many false positives that match the quotients. On the other hand, LD-Sketch uses associative arrays to keep track of frequent items, and have recall exactly equal to 100%. Its precision is as low as 80% (when $\phi = 1$MB). However, we can increase its precision with distributed detection (see below).

Experiment 2 (Accuracy for distributed detection of LD-Sketch). We now study the accuracy of LD-Sketch using distributed detection. We deploy LD-Sketch in $q = 5$ workers, each of which is configured with $\frac{M}{q}$ counters for heavy hitter detection and $\frac{2M}{q}$ for heavy changer detection, where $M = 10^6$ as in Experiment 1. We fix $\gamma = 0$ and vary both d and ϕ. Other parameters are the same as in Experiment 1.

Figure 5 shows the results of different values of d. When $d = 1$, the result is similar to the local detection in Experiment 1. When $d > 1$, the heavy hitters can be detected with 100% recall and precision. For heavy changers, choosing $d = 2$ improves the precision of local detection algorithm from 76% to 97% for threshold 1MB, and choosing $d = 3$ can even achieve 100% of precision. However, false negatives will be introduced when $d > 1$ since we may not partition the data items of a key evenly across the workers. Nevertheless, our evaluation shows that the false negative rate is less than 2%, which we believe is acceptable in practice.



(a) Single worker



(b) Multiple workers

Fig. 6. Experiment 3 (Update throughput).

Experiment 3 (Update throughput). Finally, we measure the update throughput using a single worker (local detection) and multiple workers (distributed detection). The parameters are same as in Experiment 1.

Figure 6(a) presents the update throughput versus the threshold ϕ when a single worker is used. The throughput changes marginally as the threshold grows. Note that the main bottleneck of the sketch-based algorithms is the hash operations, and the number of hash operations remains the same as it depends on the fixed parameter δ. Thus, the throughput is similar for all ϕ. From the figure, SeqHash is the slowest because it employs multiple sketches, in which the hash operations slow down the overall throughput. LD-Sketch, CGT and Fast Sketch perform the same number of hash operations, but LD-Sketch requires more time to update the associative array. When $\phi = 1$MB, it archives only 72% and 63% the throughput of CGT and Fast Sketch, respectively.

Nevertheless, we can boost the throughput of LD-Sketch via parallelization. Figure 6(b) shows the update throughput of LD-Sketch using multiple workers. We see that the throughput increases almost linearly as the number of worker grows. When using five workers, LD-Sketch can reach over 20Gb/s of throughput. This shows the scalability of LD-Sketch in a distributed architecture.

Summary of results. LD-Sketch achieves higher accuracy than many other sketch-based algorithms for heavy key detection. A trade-off is that the speed of LD-Sketch is slightly slower than other sketch-based algorithms. Using distributed detection improves the precision of LD-Sketch significantly while losing no more than 2% recall, and in the meantime, increases the update throughput.

VI. Related Work

Counter-based techniques. The Misra-Gries algorithm [18] maintains an associative array of counters and keeps tracks of frequent items (see Section II-B). Lossy Counting [16] extends the Misra-Gries algorithm by tracking the estimation error for each key. Space-Saving [17] improves the time and space efficiency with some specialized data structure. Probabilistic Lossy Counting [10] provides probabilistic guarantees on accuracy and consumes less memory by allowing errors in Lossy Counting. The above algorithms only work for heavy hitter detection, but do not address heavy changer detection.

Sketch-based techniques. Count-Sketch [3], [5] and Count-Min [9] are two well-known sketch algorithms for frequency estimation. The two algorithms differ in space requirement and error bounds. A comprehensive study [6] shows that both algorithms have similar performance in practice. Sketches have been used in heavy hitter detection (e.g., [11]) and heavy changer detection (e.g., [13]).

Some studies propose to efficiently restore heavy keys. Cormode et al. [7] consider a hierarchical structure of keys and formalize the problem of finding hierarchical heavy hitters. Deltoids [8] exploits group testing to construct heavy keys from the extra information of buckets. Reversible Sketch [20] generalizes this approach by searching smaller divided keys and constructing heavy keys from the searched results. SeqHash [2] constructs a sequence of subspaces and searches each subspace based on the results of prior ones. Fast Sketch [14] optimizes the space and time complexity of group testing by a quotient technique. Group testing assumes that there is exactly one heavy key in a bucket. It leads to a high false negative rate when multiple heavy keys are hashed to the same bucket due to constrained memory, as shown in our evaluation. In contrast, LD-Sketch queries the heavy key candidates in the buckets and guarantees no false negatives.

Distributed detection. Cormode et al. [4] exploit the linear property of sketches for distributed detection, and reduce I/O using a prediction model. Manjhi et al. [15] organize all workers into a tree structure, in which each level has its own error parameter to control the detection accuracy. Yi et al. [22] address the worst-case communication complexity in distributed streaming. However, the above studies only address heavy hitter detection, while the distributed detection of LD-Sketch addresses heavy changer detection as well.

VII. Conclusions

We present LD-Sketch, a novel distributed sketching design that aims to achieve real-time detection of traffic anomalies, including heavy hitters and heavy changers, in today's IP networks. It combines the classical counter-based and sketch-based techniques, and leverages parallelization of the emerging distributed streaming architectures to achieve both accurate and scalable detection. It is composed of local detection and distributed detection, and for both components we derive the error bounds, space complexity, and time complexity. We show via extensive trace-driven experiments that LD-Sketch provides more accurate detection than existing sketch-based techniques, and has both accuracy and scalability improvements in a distributed setting.

Acknowledgments

This work was supported by (i) GRF CUHK413711 from the Research Grant Council of Hong Kong and (ii) TH127485 from Huawei Technologies.

References

[1] Apache Flume. http://flume.apache.org.

[2] T. Bu, J. Cao, A. Chen, and P. P. Lee. Sequential Hashing: A Flexible Approach for Unveiling Significant Patterns in High Speed Networks. *Computer Networks*, 54(18):3309–3326, 2010.

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

[4] G. Cormode and M. Garofalakis. Sketching Streams Through the Net: Distributed Approximate Query Tracking. In *Proc. of VLDB*, 2005.

[5] G. Cormode, M. Garofalakis, P. J. Haas, and C. Jermaine. *Synopses for Massive Data: Samples, Histograms, Wavelets, Sketches*. Now Publishers Inc., Jan. 2012.

[6] G. Cormode and M. Hadjieleftheriou. Methods for Finding Frequent Items in Data Streams. *VLDB Journal*, 19(1):3–20, 2010.

[7] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava. Finding Hierarchical Heavy Hitters in Data Streams. In *Proc. of VLDB*, 2003.

[8] G. Cormode and S. Muthukrishnan. What's New: Finding Significant Differences in Network Data Streams. In *Proc. of INFOCOM*, 2004.

[9] G. Cormode and S. Muthukrishnan. An Improved Data Stream Summary: The Count-Min Sketch and its Applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[10] X. Dimitropoulos, P. Hurley, and A. Kind. Probabilistic Lossy Counting: An Efficient Algorithm for Finding Heavy Hitters. *SIGCOMM Comput. Commun. Rev.*, 38(1):5–5, Jan. 2008.

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

[12] X. He, P. P. C. Lee, L. Pan, C. He, and J. C. S. Lui. A panoramic view of 3G data/control-plane traffic: Mobile device perspective. In *Proc. of IFIP/TC6 Networking*, 2012.

[13] B. Krishnamurthy, S. Sen, Y. Zhang, F. Park, and Y. Chen. Sketch-based Change Detection : Methods , Evaluation , and Applications. In *Proc. of IMC*, 2003.

[14] Y. Liu, W. Chen, and Y. Guan. A Fast Sketch for Aggregate Queries over High-Speed Network Traffic. In *Proc. of INFOCOM*, 2012.

[15] A. Manjhi, V. Shkapenyuk, K. Dhamdhere, and C. Olston. Finding (Recently) Frequent Items in Distributed Data Streams. In *Proc. of ICDE*, 2005.

[16] G. S. Manku and R. Motwani. Approximate Frequency Counts over Data Streams. In *Proc. of VLDB*, 2002.

[17] A. Metwally, D. Agrawal, and A. E. Abbadi. Efficient Computation of Frequent and Top-k Elements in Data Streams . In *Proc. of ICDT*, 2005.

[18] J. Misra and D. Gries. Finding Repeated Elements. In *Science of Comput Program 2*, 1982.

[19] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari. S4: Distributed Stream Computing Platform. In *KDCloud*, Dec. 2010.

[20] R. Schweller, Z. Li, Y. Chen, Y. Gao, A. Gupta, Y. Zhang, P. Dinda, M. Y. Kao, and G. Memik. Reversible Sketches: Enabling Monitoring and Analysis over High-Speed Data Streams. *IEEE/ACM Transactions on Networking*, 15(5):1059–1072, 2007.

[21] Storm. https://github.com/nathanmarz/storm.

[22] K. Yi and Q. Zhang. Optimal Tracking of Distributed Heavy Hitters and Quantiles. In *Proceedings of PODS*, 2009.