Crossroads: A Practical Data Sketching Solution for Mining Intersection of Streams*

Zhenglin Yu
Georgia Tech
Atlanta, GA, USA
yzhenglin@gatech.edu

Zihui Ge
AT&T Labs - Research
Florham Park, NJ, USA
gezihui@research.att.com

Ashwin Lall
Denison University
Granville, OH, USA
lalla@denison.edu

Jia Wang
AT&T Labs - Research
Florham Park, NJ, USA
jiawang@research.att.com

Jun (Jim) Xu
Georgia Tech
Atlanta, GA, USA
jx@cc.gatech.edu

He Yan
AT&T Labs - Research
Florham Park, NJ, USA
yanhe@research.att.com

ABSTRACT

The explosive increase in cellular network traffic, users, and applications, as well as the corresponding shifts in user expectations, has created heavy needs and demands on cellular data providers. In this paper we address one such need: mining the logs of cellular voice and data traffic to rapidly detect network performance anomalies and other events of interest. The core challenge in solving this problem is the issue that it is impossible to predict beforehand where in the traffic the event may appear, requiring us to be able to query arbitrary subsets of the network traffic (e.g., longer than usual round-trip times for users in a specific urban area to connect to FunContent.com using a particular model of phone). Since it is infeasible to store all combinations of such data, especially when it is collected in real-time, we need to be able to summarize the traffic data using succinct sketch data structures to answer these queries.

The major contribution of this paper is the introduction of a scheme, called Crossroads, that can be used to compute the intersection of the measurements between two overlapping streams. For instance, in the above example, it is possible to compute the intersection of all the data going between the downtown area and FunContent.com with all the data generated by the model of phone to detect anomalous RTT behavior. In effect, this gives us a way to essentially "square root" the number of sketches that we need to maintain, transforming a prohibitively expensive problem to one that is tractable in practice. We provide rigorous analysis of our sketch and the trade-offs between memory footprint and accuracy. We also demonstrate the efficacy of our solution via simulation on data collected at a major cellular service carrier in the US.

*This work has been supported in part by the collaborative NSF project that includes awards CNS 1218092 and CNS 1217758.

IMC'14, November 5–7, 2014, Vancouver, BC, Canada.

http://dx.doi.org/10.1145/2663716.2663733.

Categories and Subject Descriptors

C.2.3 [**Computer-Communication Networks**]: Network monitoring; C.4 [**Performance of Systems**]: Measurement techniques

General Terms

Algorithms; Networking

Keywords

Traffic analysis; Algorithms; Data Streaming

1. INTRODUCTION

The past several years have seen an explosive growth of mobile network technologies and services in all conceivable dimensions. The amount of cellular network traffic has dramatically increased year over year. Today a large percentage of the population on this planet – especially young people – spend a large chunk of their daily lives on mobile devices such as smartphones and tablets. More and more mobile services are provided through smartphone apps, allowing people to send/receive emails and text messages, watch videos, or play online games using their smartphones.

However, this growth also poses a significant challenge to cellular network providers in that it makes the performance anomalies in cellular networks and services hard to measure, detect, and diagnose. Cellular service providers do have sophisticated monitoring of network elements for faults and performance. However, services have gotten dramatically more complicated over recent years, with a plethora of sophisticated devices (phones, tablets) and a tremendous variety of apps and online services. Knowing when a single type of device app or device/app combination is having performance problems requires comprehensive service monitoring and sophisticated problem detection and trouble isolation, and yet is critical to being able to rapidly resolve such issues. Given the complexity of this network ecosystem, issues may be introduced by the device (e.g., software), network, application server, or by some complex interaction between different combinations of these. So simply monitoring the network or overall service performance will not detect all possible issues.

1.1 Problem Statement

In this work, we identify a challenging data mining problem that is critical for automatically detecting anomalies and degradation of performance, and propose a data sketching solution for it. The type of event we are looking for in this network environment is a set of cellular calls or data sessions whose performance is negatively impacted by a common cause. An example event could be that more than 100 sessions/calls (say during a five-minute measurement interval) from customers in a "Metropolis" (a fictitious city name) downtown location to FunContent server using "Magic Phone 7" (a fictitious device name) that run on "Magic OS 88.8" (a fictitious OS name) have longer-than-usual round-trip times (RTT) during a 5-minute time window τ. In this case, network operators will be asked to look into possible root cause(s) of the event which could include Magic Phone 7 operating systems software issues (e.g., the "Magic OS" version is too recent for the old model phone), routing problems along the path from that Metropolis location to San Francisco bay area (where Facebook servers are located), or software problems with the FunContent app or at the server side (e.g., incompatibility with an old version of the app used by many Metropolis-area customers).

If we were able to record every cellular packet and store it in a traditional static database, identifying such performance anomaly events becomes a variant of the association-rule mining problem. For example, our task would be looking for the following association rule in the abovementioned scenario:

"Device = Magic Phone 7" &
"OS = Magic OS 88.8" &
"Application = FunContent.app" &
"Source = Metropolis downtown location" &
"Destination = FunContent.com" &
"Time = τ"
$\Rightarrow$ "unusually long RTT".

Our problem here is much more challenging because in a major cellular network the raw traffic comes in too fast to be saved into a database and is too large to be stored for an extended period of time. In other words, we need to identify such association rules on the fly while the packets stream in. For the "streaming version" of the abovementioned problem, a naive solution is to maintain, for every possible value combination of the attributes "Device", "OS", "Application", "Source", "Destination", and "Time", a few statistics counters for tracking the average RTT of the set of packets that matches the value combination. While this solution may barely work today, when the total number of possible value combinations of these attributes is still in the tens or hundreds of millions, it will not work in the near future, as the explosive growth in the number of applications, devices, and content providers (destinations) will soon drive the total number of such value combinations to billions or even trillions, making it impossible to fit all these counters in main memory (DRAM). While disks (or solid state drives) are still large enough to store complete summaries, they are too slow to allow for per-packet updates (to the counters) in such a high-speed cellular network.

In this work, we propose a general memory-efficient technique for performing association-rule mining over high-speed data streams, of which the abovementioned scenario is a specific case. Data streaming is concerned with processing a long stream of data items in one pass using a small working memory in order to approximately answer a query regarding the stream. The key challenge is to "remember," in this small memory, as much information pertinent to the query as possible. Such information is often organized as a synopsis data structure referred to as a *sketch* and different sketch data structures are usually needed to answer different types of queries over the data stream. These sketches are inherently lossy, greatly reducing the storage cost of the data at the cost of some accuracy – there are well-understood analytical tradeoffs between the space savings and accuracy of such sketches. Our solution reduces the amount of working space needed to (approximately) track all possible attribute value combinations (say there are N of them) from $O(N)$ to $O(\sqrt{N})$, without significantly degrading the monitoring accuracy for major performance anomalies. This allows the working space to reside entirely in the main memory (DRAM), and thereby allows for the processing of each and every data item (in this case a packet) in the stream – and the updating of the corresponding sketches – on the fly.

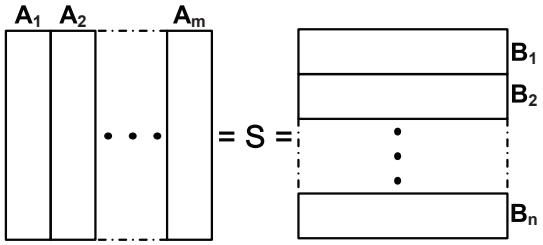


Figure 1: Two different partition scenarios of the same set. We partition the identical set S as subsets A_1, A_2... A_m and as subsets B_1, B_2... B_n

1.2 The "Crossroads" Approach

Our solution consists of three steps. The first step is to partition the set of attributes that needs to be mined for associations into two subsets so that the number of value combinations of attributes in each subset is much smaller than – and ideally close to the square root of – that of all attributes. As shown in Fig. 1, each subset of attributes can be viewed as a dimension along which the data set S (in our case consisting of all packets in a cellular network) is partitioned. For example, in the abovementioned scenario, one dimension could include all possible "Source" and "Destination" combinations, and the other "Device type", "OS", and "Application" (including its version number) combinations; while the number of joint value combinations of all these attributes in the future could be in trillions ($\sim 10^{12}$), that of each subset may only be in millions ($\sim 10^6$). In this way, the data set S can be partitioned into (data) subsets A_i for $i = 1, 2, ..., m$ along one dimension, each of which corresponds to the subset of data that takes a certain value combination along that dimension, and similarly into (data) subsets B_j for $j = 1, 2, ..., n$ along the other. In the abovementioned application scenario, the set of packets that match ("Source = the Metropolis downtown location" & "Destination = FunContent.com") could be one such A_i, and the set of packets that match ("Device = Magic Phone 7" & "OS = Magic OS 88.8" & "Application = FunContent.app" & "Time = τ") could be one such B_j. Then the set of packets that match the filter ("Device = Magic Phone 7" & "OS = Magic

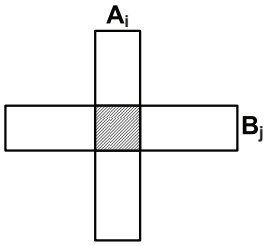


Figure 2: We can compute functions on the intersection of arbitrary sets A_i and B_j

OS 88.8" & "Application = FunContent.app" & "Source = the Metropolis downtown location" & "Destination = FunContent.com" & "Time = τ") – of which we would like to measure and monitor the performance metrics (e.g., average RTT) – is precisely the intersection of A_i and B_j, as illustrated in Fig. 2. As will become clear shortly, we will estimate these performance metrics by taking such intersections, and hence the name "Crossroads."

In the second step, for each set A_i, a sketch is constructed to succinctly summarize the performance metrics (e.g., average RTT) of all data items in it. The same is done for each B_j. We will talk about the sketch data structures suitable for this purpose later in Section 3. The novelty of this approach lies mainly in the third step, which is to derive the performance metrics of the data inside each $A_i \cap B_j$ by intersecting the sketches constructed for A_i and that constructed for B_j.

The potential further loss of accuracy in this intersection process aside, one may be tempted to take intersection on three or more such sketches. This is however provably impossible, if the Tug-of-War sketch is used as the underlying sketch data structure [1]. The Tug-of-War sketch is in general preferred over other types of sketches due to its lower space and computational complexities, and the much better numerical stability of the Gaussian distribution used by it. Slightly disappointing as this impossibility result might be, it serves as a somber reminder that we should count our blessings that we are able to perform even the two-way intersection after all.

It is assumed, in the above example, that the full data set is available to the association-rule mining (ARM) algorithm – that is, all attribute value combinations are stored. The contribution of this work is to significantly reduce the amount of data stored so that it can be performed over a high-speed data stream. We do not propose any new ARM algorithms, but give a way to reduce the amount of data that such an algorithm would need to use. Although we believe our technique could be used to reduce the space-complexity of many ARM algorithms, that is outside the scope of this paper.

2. PRELIMINARIES

In this section, we give a formal description of the problem and illustrate it with some examples.

For the problem of detecting anomalies and other events in a cellular network, we would like to estimate some useful quantities such as the *moments* of some important metrics (e.g., RTT) of the data stream. The kth moment of a packet stream is defined as: $M_k = \sum_i v_i^k$, where v_i is the value (e.g., RTT) of the ith packet in a stream. In particular, the moments $M_0 = \sum_i v_i^0 = \sum_i 1$ and $M_1(v) = \sum_i v_i$ give the number of packets and the sum of such values in all the packets in the stream, respectively. These quantities together allow us to estimate the mean of such values in the stream. The second moment ($M_2 = \sum_i v_i^2$) is even more interesting, since, normalized by the total number of packets M_0 in the stream, it can be decomposed into the sum of two quantities: (1) the square of the empirical mean v value, and (2) the empirical variance of the v values, across all packets in the stream (i.e., $\frac{M_2(v)}{M_0(v)} = E(v)^2 + Var(v)$). Any significant increase in either the mean or the variance, both constituting a performance anomaly, will be reflected in the second moment.

Here is a simple example of a stream containing 3 different packets. Let the RTT values of the 3 packets be 50, 100, 100. For this stream, the number of distinct elements will be $M_0(RTT) = 3$, the sum of RTT values in the stream will be $M_1(RTT) = 50 + 100 + 100 = 250$, and the second moment will be $M_2(RTT) = 50^2 + 100^2 + 100^2 = 22,500$.

Note that in the data streaming literature, we are usually concerned with the *frequency moments* of the stream, that is, moments based on the frequency with which each unique identifier appears. In the networking context, this identifier is usually that of a transport-layer (TCP or UDP) flow, i.e., the five-tuple consisting of source and destination IP addresses and port numbers, and the protocol. In our case, however, we consider every packet to have a unique identifier, or to be a singleton "flow" itself. In other words, we are interested in the much easier-to-compute moments of the (RTT) values of individual singleton flows. For a single stream this can trivially be computed using a single counter that is incremented with each new (RTT) value. The reason why our problem is challenging, and deserves this effort, is that we are interested in efficiently computing the moments of the *intersection* of two massive data streams. More precisely, we would like to compute the moments of the (RTT) values associated with the set of data items that appear in both data streams (i.e., their "intersection").

3. INTERSECTION ALGORITHM

In this section, we first describe the algorithm for sketching the data. We then explain how our intersection algorithm can be used for estimating the second frequency moment of a traffic feature. We use RTT as a concrete example, though we emphasize that any numeric feature can be used. Next, we formalize our M_2 intersection algorithm and provide theoretical guarantees and rigorous statistical analysis for it. Finally, we describe the design of our M_0 and M_1 algorithms and explain how they work. Once again, the sketch data structures are not novel, but the way in which they are used for computing intersections is the main contribution of our work.

3.1 Sketching Algorithm

As explained before and shown in Fig. 2, we would like to be able to estimate certain metrics (e.g., RTT) associated with data items that belong to the intersection of any A_i, $i = 1, 2, ..., m$ with any B_j, $j = 1, 2, ..., n$. We do so by maintaining a total of $m+n$ sketches, namely, one for each set A_i, $i = 1, 2, ..., m$ and one for each set B_j, $j = 1, 2, ..., n$. In our application scenario, the data are simply a massive stream of cellular network packets. We identify four important at-

tributes I, J, V, and ID in each packet. The abovementioned partitioning of the whole stream into $A_i's$ and $B_j's$ is based on the value a packet takes on attribute I and J respectively. The V attribute is the metric (say RTT) that is of interest to us. The ID attribute is essentially the set of bits in a packet that uniquely identifies the packet. We emphasize this ID attribute is not the flow identifier (the abovementioned five tuple) although the latter could be a part of the former, since we consider each packet a "singleton flow" for the purpose of the second moment estimation, as we explained earlier.

Let us denote the sketches for A_i and B_j as $S(A_i)$ and $S(B_j)$, respectively, and let $M(v)$ be a function that computes the moments for a certain value v (e.g., $M(v) = v^2$ for M_2). Then our goal is to estimate $\sum_{pkt \in A_i \cap B_j} M(pkt.V)$, which denotes the moment of the values of all packets in the intersection of the sets A_i and B_j, from the sketches $S(A_i)$ and $S(B_j)$. Therefore, as shown in Algorithm 1, with the arrival of each packet pkt, we need to update both sketches $S(A_{pkt.I})$ and $S(B_{pkt.J})$ with its contribution $pkt.V$. We will explain in the next section how this contribution to both sketches are "linked" by the identifier of the packet $pkt.ID$ so that it can be accounted for and later retrieved from their intersection.

Algorithm 1 Processing algorithm

UPDATE:

1: Upon the arrival of a packet pkt
2: v:=pkt.V
3: i:=pkt.I
4: j:=pkt.J
5: Update(S(A_i), v, pkt.ID)
6: Update(S(B_j), v, pkt.ID)

As conventional data sketches weren't designed to handle such intersection operations, our work is focused on how to do that. We choose the *Tug-of-War sketch* as the underlying sketch data structure and build the intersection capability on top of it. Alon et al. [2] proposed this remarkable sketch, later named the "tug-of-war" sketch in [1], for estimating the second moment of a data stream using only logarithmic space. For all $\epsilon, \lambda > 0$, the tug-of-war sketch uses only $O(\frac{\log(1/\epsilon)}{\lambda^2})$ counters to sketch the second moment of a stream with the guarantee that the probability with which the relative error is larger than λ is at most ϵ. The sketch allows arbitrary and efficient updates for the values. Let $\overrightarrow{s} = \{s_1, s_2, ...s_n\}$ be a vector of length n, where each entry s_i takes on value $+1$ or -1 with equal probability $\frac{1}{2}$, and $\overrightarrow{v} = \{v_1, v_2, ...v_n\}$ where $\{v_1, v_2, ...v_n\}$ are the respective V values of the n packets in the stream. Then the value of a counter in the sketch is $\overrightarrow{v} \cdot \overrightarrow{s} = v_1 s_1 + v_2 s_2 + \ldots + v_n s_n$ [2]. It was shown in [2] that the square of this counter value is an unbiased estimator for the second moment of the v_i values, and a small number of such estimators (i.e., squares of the values in multiple counters) can be combined to provide an accurate estimation of the second moment, as characterized by the abovementioned space-accuracy tradeoff.

One key property of the tug-of-war sketch is that if two sketches have the same size and use the same vector $\overrightarrow{s}$, then we can add up the two sketches to get the direct sum which is equivalent to taking the union of the original streams. This feature of allowing the computation of the union of two sketches is a vital property that we will later leverage in our algorithm design.

Algorithm 2 The Tug-of-War algorithm

PRE-PROCESSING:

1: Fix K independent hash functions $h_k : \boldsymbol{U} \rightarrow \{+1, -1\}$, $k = 1, 2, ..., K$, each of which is four-wise independent
2: Initialize $Z[k] := 0$ for $k = 1, 2, ..., K$

ALGORITHM:

1: Upon the arrival of a packet pkt
2: For $k := 1$ to K
3: $Z[k] := Z[k] + pkt.V * h_k(pkt.ID)$
4: Return $\left(\sum_{k=1}^{K} Z[k]^2\right)/K$

3.2 Intersection algorithm for estimating M_2

Recall that our data are a long stream of packets, and for each packet pkt, we identify four important attributes I, J, V, and ID. In this section, we explain how we compute the M_2 of the V values (RTTs in this case) for the intersection of a certain pair of sub-streams A_i and B_j. The sub-stream A_i consists of all packets whose I attribute value is equal to i, and the sub-stream B_j consists of all packets whose J attribute value is equal to j. To simplify the notation, we drop the subscripts in A_i and B_j in the sequel, referring to them as A and B respectively. With the streaming context of A and B being clear, we will call them sets rather than sub-streams in the sequel.

3.2.1 Algorithm Overview

We use the tug-of-war technique to sketch both A and B. Each sketch – $S(A)$ or $S(B)$ – is made up of K counters $\{Z[k]\}$, $k = 1, 2, ..., K$. For convenience of notation, we will write $S(A)$ or $S(B)$ as S_A or S_B in the sequel. Each counter $Z[k]$ is paired with a hash function h_k that generates the above-mentioned random vector $\overrightarrow{s}$ (now called $\overrightarrow{s_k}$ to distinguish them from one another) with which the V values of the packets in the stream, viewed as a vector, will take an inner product. The "Update(S(*), v, pkt.ID)" procedure inside Algorithm 1, when the underlying sketch is the tug-of-war, is shown in Algorithm 2. Upon the arrival of a packet pkt that belongs to the sub-stream, the update to to the k^{th} counter is simply $Z[k] := Z[k] + pkt.V * h_k(pkt.ID)$, where each h_k is a hash function that deterministically outputs $+1$ or -1 with equal probability given any input value. As mentioned earlier, there is a tradeoff between the accuracy of the estimation and the number of counters K used. When we refer to the size of the sketch, we mean the number of counters K.

The tug-of-war technique has a useful property that, when two tug-of-war sketches $S(A)$ and $S(B)$ employ the same number of counters and set of hash functions, $S_{A \bigcup B}$, the sketch for the union of two sets, is simply the component-wise addition of S_A and S_B (i.e., their corresponding counters added up). Making use of this property by doing exactly so, we define $S_{A \oplus B}$ as the component-wise sum of S_A and S_B. Our estimation of the M_2 of $A \bigcap B$ is simply $\widehat{M_2}(S_{A \bigcap B}) = \frac{1}{2} \cdot \{M_2(S_{A \oplus B}) - M_2(S_A) - M_2(S_B)\}$, where $M_2(S_{A \oplus B})$, $M_2(S_A)$, and $M_2(S_B)$ can be obtained using the standard estimation procedures for the tug-of-war sketch.

3.2.2 Theoretical Guarantees

In this section, we derive the variance of the estimator $\widehat{M_2}(S_{A \bigcap B})$ when only one (instead of K) hash function and counter is used in sketching. We define two parameters $\gamma_A, \gamma_B \leq 1$ as $\sum_{A\cap B} v_i^2 = \gamma_A \sum_A v_i^2$ and $\sum_{A\cap B} v_i^2 = \gamma_B \sum_B v_i^2$. In other words, γ_A and γ_B represent the proportion of the intersection part respectively in set A and set B. Intuitively, this is the amount of overlap the intersection has with either set and we will need it to be large enough because otherwise the "signal" (M_2 of the intersection) will be drowned out by the noise (errors in estimating $M_2's$ of A and B from their respective sketches). The following theorem demonstrates that such an estimator is unbiased and has reasonably low variance. Note this variance can be further reduced by about a factor of K, when K counters (and hash functions) are used.

Theorem 1 *The M_2 Intersection Algorithm produces an unbiased estimator for $M_2(S_{A \bigcap B})$, and the variance of the estimator is* $(\sum_{A\cap B} v_i^2)^2(\frac{1}{\gamma_A\gamma_B} - 1)$

Proof Let $\widehat{Y}$ denote the estimator for $M_2(S_{A \bigcap B})$ and recall that v_i denotes the value for the element with ID i. The estimator for the sketch S_A is $\widehat{M_2}(S_A) = (\sum_A v_i s_i)^2$ and that for sketch S_B is $\widehat{M_2}(S_B) = (\sum_B v_i s_i)^2$. In both sketches, every choice of $s_i \in \{+1, -1\}$ is identical as both use the same hash function $h : \boldsymbol{U} \to \{+1, -1\}$. The estimator $\widehat{M_2}(S_{A\oplus B})$ for the second frequency moment of $S_{A\oplus B}$ is then $\widehat{M_2}(S_{A\oplus B}) = (\sum_A v_i s_i + \sum_B v_i s_i)^2$.

Now we prove that our algorithm can produce an unbiased estimator for $M_2(S_{A\cap B})$. From the above algorithm, our estimator is:

$$\begin{aligned}
\widehat{Y} &= \frac{1}{2} \cdot \{\widehat{M_2}(S_{A\oplus B}) - \widehat{M_2}(S_A) - \widehat{M_2}(S_B)\} \\
&= \frac{1}{2} \cdot \{(\sum_A v_i s_i + \sum_B v_i s_i)^2 \\
&\quad -(\sum_A v_i s_i)^2 - (\sum_B v_i s_i)^2\} \\
&= (\sum_A v_i s_i)(\sum_B v_i s_i).
\end{aligned}$$

Then, the expected value of this estimator is

$$\begin{aligned}
E[\widehat{Y}] &= E[(\sum_A v_i s_i)(\sum_B v_i s_i)] \\
&= E[\sum_{A\cap B} v_i^2 s_i^2 + \sum_{i\in A-B, j\in B-A} v_i s_i v_j s_j + \\
&\quad \sum_{i\in A\cap B, j\in B-A} v_i s_i v_j s_j + \sum_{i\in A-B, j\in A\cap B} v_i s_i v_j s_j] \\
&= \sum_{A\cap B} v_i^2 = M_2(S_{A \bigcap B}).
\end{aligned}$$

In the second equality, for all $i \neq j$ we know that s_i and s_j are independent, so $E[s_i s_j] = E[s_i] \cdot E[s_j] = 0$ and all but the first term disappear. Also note that $s_i^2 = 1$ as each $s_i \in \{1, -1\}$, leading to the simplification of the first term. Thus, $\widehat{Y}$ is an unbiased estimator for $M_2(S_{A\cap B})$.

Now we will analyze the variance of our estimator in the algorithm. First, we compute

$$\begin{aligned}
E[\widehat{Y}^2] &= E[(\sum_A v_i s_i)^2 \cdot (\sum_B v_i s_i)^2] \\
&= E[(\sum_{A\cap B} v_i s_i + \sum_{A-B} v_i s_i)^2 \\
&\quad \cdot(\sum_{A\cap B} v_i s_i + \sum_{B-A} v_i s_i)^2] \\
&= \sum_{A\cap B} v_i^4 + 2 \sum_{A\cap B, i<j} v_i^2 v_j^2 + (\sum_{A\cap B} v_i^2)(\sum_{B-A} v_i^2) \\
&\quad +(\sum_{A\cap B} v_i^2)(\sum_{A-B} v_i^2) + (\sum_{A-B} v_i^2)(\sum_{B-A} v_i^2).
\end{aligned}$$

In the last equality we once again drop terms that include $s_i s_j$ where $i \neq j$ as their expectation is equal to 0.

From the derivation of the expectation, we know that:

$$(E[\widehat{Y}])^2 = \textstyle\sum_{A\cap B} v_i^4 + 2\sum_{A\cap B, i<j} v_i^2 v_j^2.$$

so the variance of the estimator is

$$\begin{aligned}
Var[\widehat{Y}] &= E[\widehat{Y}^2] - (E[\widehat{Y}])^2 \\
&= (\sum_{A\cap B} v_i^2)(\sum_{B-A} v_i^2) + (\sum_{A\cap B} v_i^2)(\sum_{A-B} v_i^2) \\
&\quad +(\sum_{A-B} v_i^2)(\sum_{B-A} v_i^2) \\
&= (\sum_{A\cap B} v_i^2)^2(\frac{1}{\gamma_B} - 1) + (\sum_{A\cap B} v_i^2)^2(\frac{1}{\gamma_A} - 1) \\
&\quad +(\sum_{A\cap B} v_i^2)^2(\frac{1}{\gamma_A} - 1)(\frac{1}{\gamma_B} - 1) \\
&= (\sum_{A\cap B} v_i^2)^2(\frac{1}{\gamma_A\gamma_B} - 1).
\end{aligned}$$

Numerical Illustration: For instance, assume that the intersection of the two streams only has 1% of the value of both A and B (i.e., $\gamma_A = \gamma_B = 0.01$). Then, if we use just 100,000 counters per sketch, the variance-to-mean-square ratio is 0.1 – small enough to get under 10% error with high probability.

3.3 Intersection Algorithms for estimating M_0 and M_1

The zero-th moment M_0 indicates the number of distinct elements in a stream. Ordinarily, this quantity can be trivially tracked using just one counter for a single stream, but this trivial solution does not allow for the estimation of M_0 of the intersection of two streams. However, the above-mentioned intersection technique for estimating M_2 can be modified slightly to estimate M_0 of the intersection of two streams. We simply replace the term "*pkt*.V" in line 3 of the second routine in Algorithm 2 by the value 1. The M_0 estimator for the intersection is simply $\widehat{M_0}(S_{A \bigcap B}) = \frac{1}{2} \cdot \{M_0(S_{A\oplus B}) - M_0(S_A) - M_0(S_B)\}$. Note this simple modification works for M_0 estimation only in this special case, where every packet is a "singleton" flow.

To estimate the average RTT, we need also to estimate the first moment M_1, which is the sum of all the *RTT* values of the packets in a stream. It turns out that the M_2 intersection technique can again be slightly modified to suit our need. We simply replace the term "*pkt*.V" by its square root and the estimator remains the same as in the M_2 case.

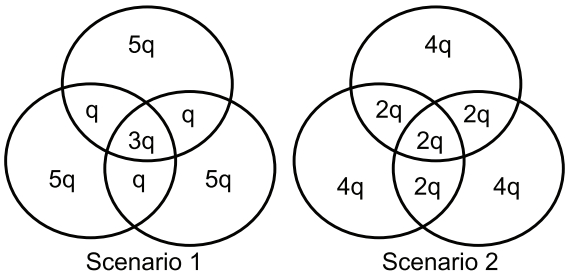


Figure 3: Two scenarios for intersection of 3 sketches.

Again, this simple modification works only in this special case, where every packet is a "singleton" flow. Otherwise, we would have to resort to much more expensive techniques such as the Cauchy distribution sketch proposed by Indyk [10]. The algorithms and analysis are very similar to the case for M_2 and are omitted here for brevity.

4. AN IMPOSSIBILITY RESULT ON 3-WAY INTERSECTION

In this section, we prove a claim stated earlier that, with Tug-of-War algorithm, it is impossible to generalize this 2-way intersection scheme to a 3-way (or higher) scheme. Note that in the Tug-of-War sketch, the final value of each counter is the result of a long random walk, which is known to be very close to Gaussian. This property will be used in the proof. Recall that our scheme uses Tug-of-War for estimating all three quantities (M_0, M_1, M_2) of interest, and is therefore governed by this impossibility result. Although it is possible to circumvent this impossibility using other types of sketches (e.g., the stable distribution sketches [10] other than the Gaussian one), they are in general much more expensive than the Tug-of-War sketch.

Intriguing as our claim is, its proof is in fact quite straightforward. Suppose three Tug-of-War sketches S_A, S_B, and S_C – each of which consists of the same number of counters – are used to summarize three data sets A, B, and C respectively. Since counter values in each sketch is i.i.d., without loss of generality it suffices to prove the case in which there is only one counter in each sketch (i.e., one random variable in each random vector). In the following, we will create two sets (of data sets) $\{A_1, B_1, C_1\}$ and $\{A_2, B_2, C_2\}$ so that the random vectors $\langle S_{A_1}, S_{B_1}, S_{C_1}\rangle$ and $\langle S_{A_2}, S_{B_2}, S_{C_2}\rangle$ have identical joint (approximately Gaussian) distributions, but $|A_1 \cap B_1 \cap C_1|$ and $|A_2 \cap B_2 \cap C_2|$ are very different in value. However, since any estimator of the 3-way intersection is necessarily a function of the joint distribution of these three sketches, this estimator cannot estimate both 3-way intersections accurately.

First consider scenario 1 in Figure 3. All three sets, A, B, and C, contain $10q$ packets each. Note that though their sizes are the same, A, B and C are distinct sets of packets. Let us assume that all the values for all packets have the same value T. In this scenario, the sketches produced from the three sets are three random variables (there is only one counter in each sketch as explained earlier) S_A, S_B, S_C. As explained before, when q is large, the marginal distribution of each random variable is Gaussian, and they have a joint Gaussian distribution. The variance of the three sketches are all $10T^2$, and their pairwise covariances $Cov[S_A, S_B]$, $Cov[S_B, S_C]$, $Cov[S_A, S_C]$ are all equal to $4qT^2$. Note that $4q$ is exactly the size of pairwise intersections $|A \cap C|$, $|A \cap B|$ and $|B \cap C|$. Thus these 3 random variables S_A, S_B, S_C have a joint Gaussian distribution with the covariance matrix:

$$\Sigma = \begin{bmatrix} 10T^2 & 4qT^2 & 4qT^2 \\ 4qT^2 & 10T^2 & 4qT^2 \\ 4qT^2 & 4qT^2 & 10T^2 \end{bmatrix}. \quad (1)$$

It can be easily verified that in scenario 2 these 3 random variables also have a joint Gaussian distribution, and have the same covariance matrix. Since the covariance matrix uniquely determines a joint Gaussian distribution, these two random vectors have the same joint probability distribution in both scenarios. However, the size of 3-way intersection in these two scenarios are $3qT^2$ and $2qT^2$ respectively. This means that there is at least 33% relative error in estimating one of these values.

5. IMPLEMENTATION ISSUES

In this section, we describe several important implementation issues with the data sketches and their cost and performance ramifications if applicable.

1. **Intersection Ratio.** We define the intersection ratio (IR) of the sets A and B as follows:

$$IR_A = \frac{size(A \cap B)}{size(A)}, IR_B = \frac{size(A \cap B)}{size(B)}, \quad (2)$$

where $size()$ denotes the number of packets contained in a set. IR_A and IR_B are equivalent respectively to γ_A and γ_B defined earlier, if the V values of all the packets are identical.

This IR has a great influence on the relative errors of our estimations, for M_0, M_1, and M_2, over the intersection $A \cap B$. For example, if $IR_A = IR_B = 0.01$, which means the intersection constitutes only 1% of the size of A or B, then these relative errors will be approximately 100 times larger than the relative errors of our estimations over A or B alone. When IR_A and IR_B are both very small (e.g., 10^{-4}), $A \cap B$ is so small compared to A and B that our approach will not produce accurate estimations over the intersection without consuming an excessive amount of counter memory. In our implementations, we dimension just enough counter memory for the cases in which IR_A and IR_B are both around or above 0.005.

2. **The ID attribute.** It has been shown that the first invariant (i.e., not including fields that are modified hop by hop by routers such as TTL) 28 bytes in an IP packet header suffice to uniquely identify a packet [21]. In the spirit of that, a similar set of bytes is used as the aforementioned ID attribute for the cellular network packets (different from IP). Our approach can also be extended to flow-level by using the unique identifier of each flow instead of each individual packet.

3. **Sketch resource allocation.** In this work, we are interested in estimating the average RTT, or equivalently $\frac{M_1(RTT)}{M_0(RTT)}$, from the underlying Tug-of-War sketches. However, since $M_0(RTT)$ is the denominator, its estimation accuracy affects the overall accuracy more than

the numerator $M_1(RTT)$. Therefore, we allocate more sketch memory, in terms of the number of counters and hash functions, for measuring M_0 than for measuring M_1.

4. **No median and all averaging.** The original Tug-of-War sketch used a median of mean of the counters [1] mainly for proving accuracy bounds rigorously. The number of different counters used is usually $s = s_1 \cdot s_2$, where s_1 represents the number of counters we average to improve the accuracy and s_2 represents the number of such averages we take the median of to improve the confidence. In this work, we decided to simply average all s counters together to arrive at our estimator, instead of also taking median, because our experiments show no improvement in empirical accuracy by doing so. Note that upon the arrival of each packet, all s counters must be updated no matter how s splits into s_1 and s_2, so unlike the bucketing issue described below, this decision does not affect the computational complexity of the per-packet update procedure.

5. **Tug-of-War bucketing.** Since the computational complexity of the update procedure of the original Tug-of-War algorithm can be quite high, especially when a large number of counters and hash functions are used, a bucketing technique first introduced in [26] is used in our implementations to reduce this complexity. Bucketing works by partitioning the available memory into several disjoint buckets of counters and introducing a new hash function. For every arriving packet, we map it to a bucket by hashing its packet ID using this new hash function. Counters in this bucket will then be updated in the same way as in the original Tug-of-War sketch. The second moment of the stream can be estimated by first obtaining the second moments of the sub-streams encoded by the buckets, and then adding them up. The computational complexity of each update can thus be reduced by a factor of the number of buckets. The accuracy loss due to bucketing is negligible, as shown in [26].

6. **Intersection overhead.** While the update of the sketches are optimized to be possible online, the intersection of pairs of sketches can be done offline as needed. There is a small computational overhead of $O(K)$ operations, where K is the sketch size used (a few thousand counters in the following experimental section). This overhead is small enough to be negligible, and well worth the savings in storage cost.

6. EVALUATION

In this section, we evaluate the accuracy and efficiency of our estimation algorithms using two types of data: real-world cellular network data from a major cellular service carrier in the US and synthetic traffic data. The cellular network data was collected during a one month period across the entire United States and was anonymized and aggregated. No personally identifiable information was gathered or used in conducting this study. We also generated synthetic data statistically similar to the real data so that we had more control over various parameters such as the sizes of the sets A and B to be intersected and the amount of overlap between them. Our results on real data show that our approach uses a memory space about 170 times smaller than the naive full-table approach (i.e., devoting one or a few statistics counters to each $A_i \bigcap B_j$) while suffering only about 10% loss in estimation accuracy.

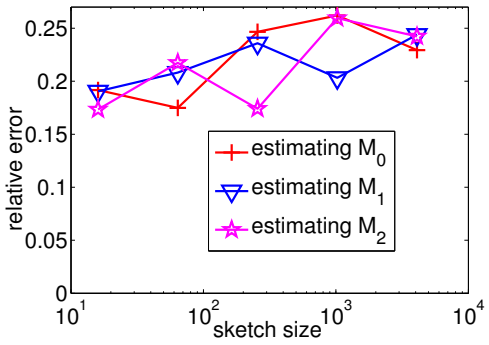


Figure 4: Synthetic results for M_2 intersection relative errors when the total memory is fixed to 4096 counters while the sketch size and the number of buckets are varied

6.1 Results for Synthetic Data

In our synthetic data, all RTTs were randomly generated values between 0 and 100 milliseconds, similar to the values we found in the real data set. For each experiment, we generated data according to whether we would like to vary the full data size (number of packets or unique combinations) or the size of the intersection compared with the size of the individual sets (i.e., intersection ratio).

We first evaluate the effect of bucketing on estimation accuracy by comparing estimation outcomes without bucketing (viewed as having one bucket containing all counters) with those with bucketing. In all cases, an identical number (4096) of counters are used. For this experiment, we fix the sizes (number of packets) of the intersecting sets A and B to be 0.1M and both intersection ratios to be 0.05. As we can see from Fig. 4, when the total number of counters we use in each sketch is set, and we vary the bucket size, defined as the number of counters in each bucket, and the number of buckets while keeping the total number of counters per sketch constant, the results do not vary substantially. This means that bucketing does not negatively impact the estimation accuracy while reducing computational complexities of the update procedure. We got similar results with various other total counter counts (not shown here in the interest of space). As there seems to be no marked performance benefit to avoid bucketing, we fixed the number of counters per bucket 16 in subsequent experiments. This means that each packet only triggers 16 counter updates, which is affordable computationally even for processing high-speed cellular network traffic.

Next, we vary the number of buckets, while fixing the bucket size to 16 (counters per bucket), to see how it affects the relative mean intersection estimation error and the relative mean sketch error in Fig. 5. Again the sizes (packets number) of both intersecting sets A and B are set to 0.1M and both intersection ratios 0.05. We found that the results for M_2, M_1 and M_0 were comparable. The reason for this is that the 3 frequency moment estimation processes all use the

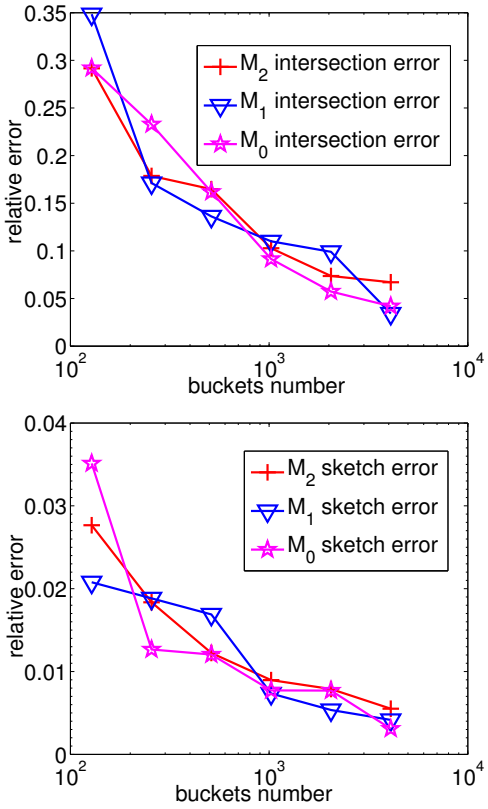


Figure 5: Synthetic results for mean intersection relative errors and sketch relative errors when varying (the number of) buckets. Bucket size set to 16 counters and the total memory usage is bucket size $\times$ buckets

same Tug-of-War sketch. In both figures, as the number of buckets gets larger, both relative sketch estimation error and intersection estimation error decrease, but with diminishing returns.

Next, we study the situation in which the intersection ratios IR_A and IR_B are varied simultaneously. The results are shown in Fig. 6. As IR increases, mean relative error decreases very quickly. The effect of increasing IR is quite notable for decreasing relative error. When IR is quite small (e.g., 10^{-5}), getting an accurate estimation for the intersection part is very difficult. As the intersection ratio gets larger, we get increasingly accurate estimates of the intersection. Note that the intersection ratio will not affect our sketch estimating errors; it affects only the intersection estimation error.

We also varied the size of A and B (with $size(A) = size(B)$) while keeping their IR constant. The results are shown in Fig. 7. We found that the mean relative error increases only slightly with the size of both sets. This suggests that the sizes of both sets do not significantly affect the relative intersection error as long as the intersection ratio remains constant. Hence, our method can be used for even larger data sets (i.e., larger packet streams) with little degradation in estimation accuracy.

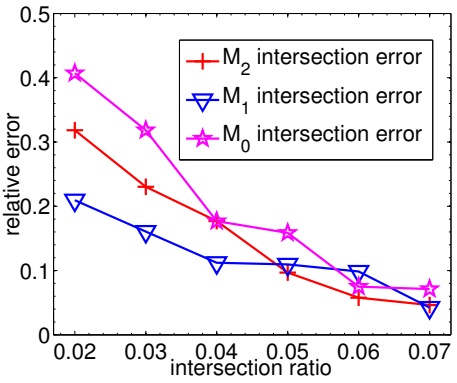


Figure 6: Synthetic results for mean intersection relative errors when varying intersection ratio. Bucket size set to 16 counters and the number of buckets to 1024

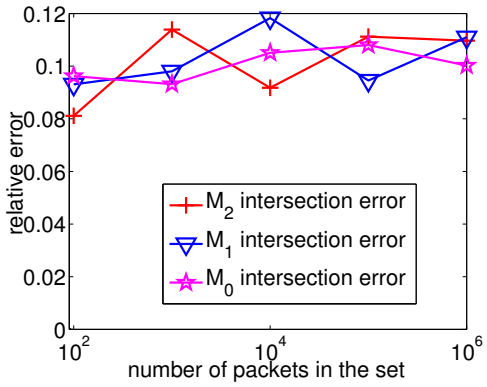


Figure 7: Synthetic results for mean intersection relative errors when varying the number of packets in the set. Bucket size set to 16 and buckets to 1024

6.2 Data Sets for Real Data

Our real-world data was collected from a major cellular service carrier in the US over the month of January 2014. It consists of anonymized TCP flow level data collected from the core network for the 3G data service provided by the cellular service carrier.

Figure 8 illustrates the architectural overview of the core network, which consists of two main types of nodes: the Serving GPRS Support Node (SGSN) and the Gateway GPRS Support Node (GGSN). The GGSN is the root node in the hierarchy of the cellular data network and responsible for sending and receiving Internet traffic to and from the cellular network. SGSN is an intermediate node that connects the lower level nodes to the GGSN through the G_n interface. Typically, a single SGSN is connected to multiple Radio Network Controllers (RNCs) and each RNC serves a geographical region through cell towers.

The raw data is collected at the G_n interface which connects the SGSNs to the GGSNs. Specifically, for each TCP flow, we first measure its end-to-end round trip time (RTT) by comparing the timestamps of IP packets during the TCP

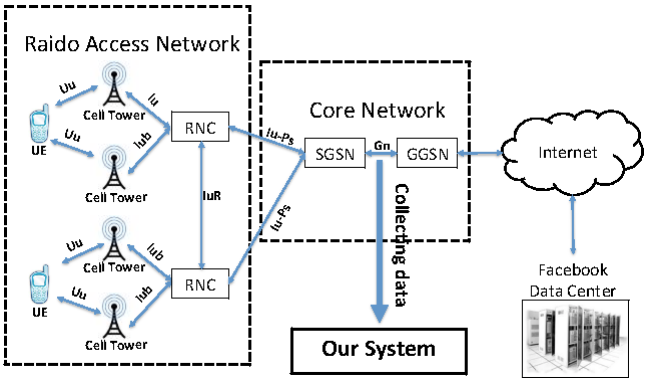


Figure 8: Data Collection Architecture

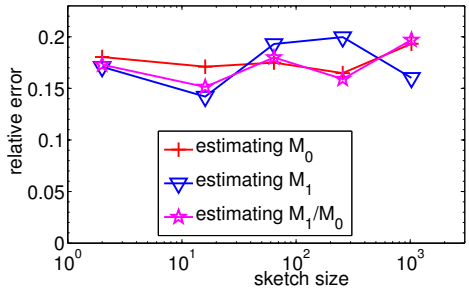


Figure 9: Results of mean intersection relative errors with fixed memory to 4096 counters and varying bucket size and the number of buckets

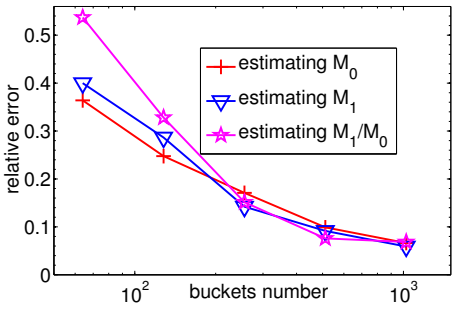


Figure 11: Results of mean intersection relative errors when varying buckets for real data. Bucket size set to 16

handshake and then associate the RTT value with various flow attributes such as standard coordinated universal time (UTC), the serving RNC that describes the user access point, the handheld device type, the application type, the content provider being accessed, etc. Due to the sheer amount of traffic volume, instead of storing the raw data for all individual flows, we only store the hourly total number of RTT measurements, the hourly summation of the RTT measurements and the hourly average RTT (computed as the sum divided by the total number) over eight different flow attributes: serving RNC, handheld device manufacturer/model, handheld device speed category, service category, content provider, day of week, hour and access point network (APN). No personally identifiable information is retained in the aggregate statistics. Being able to query these statistics for an arbitrary cross-section of the eight different flow attributes is critical for service providers to manage their service performance. In order to be able to perform a query for any arbitrary 8-tuple of these quantities, we divided these attributes into two groups. The results of some preliminary measurements showed that making serving RNC, service category, handheld device speed category and day of week as a group, and handheld device manufacturer/model, content provider, access point network and hour as another group, would minimize the total number of sketches that we would need to perform a query for an arbitrary slice of the data.

There are about 1.4 million distinct combinations for the former group described above and about 1.5 million for the latter. Thus, the cost of maintaining the value of each and every combination (a table with one counter per combination) would result in a space usage of 1.4M $\times$ 1.5M $\times$ 4 bytes, or more than 7.5 TB of storage capacity. In contrast, if we use 4096 counters in each sketch, our method needs only $(1.4M + 1.5M) \times 4096 \times 4$ bytes, or less than 45 GB. This is a savings of over two orders of magnitude that is paid for with a small loss in estimation error. Note that working with even larger data sets (for instance, monitoring a year's worth of data) will give even greater savings.

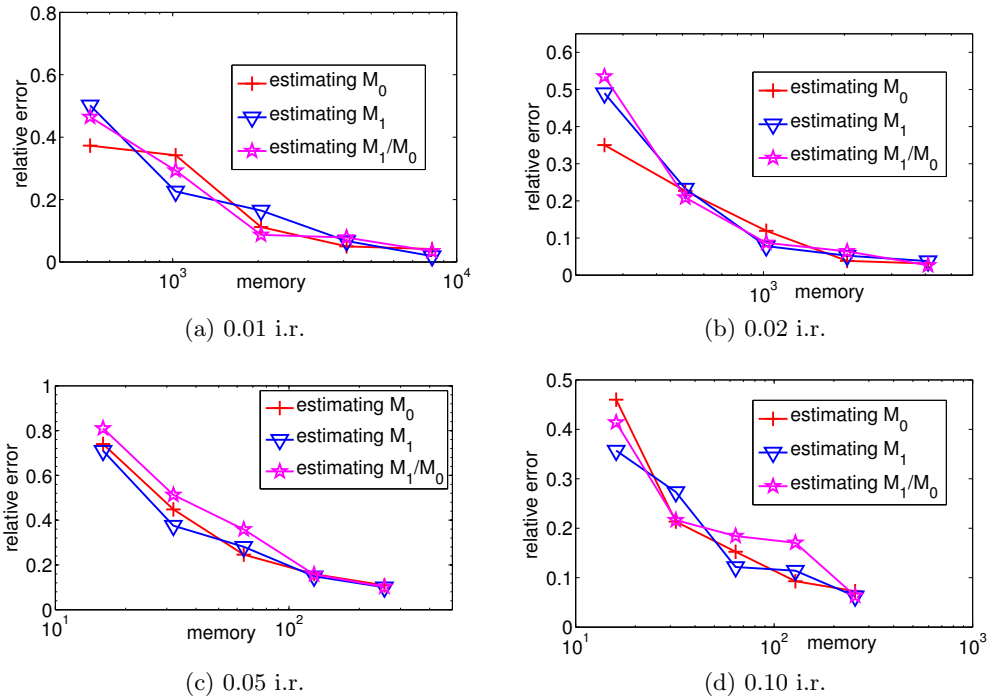
Since the data we get is already aggregated by hour, we were unable to perform per-packet processing. Instead, we updated our sketch data structures using the measurement count and measurement values for each tuple aggregated at each hour across 36 sub-regions that span the United States. A limitation of this aggregated data was that we were unable to compute the second moment values for the RTTs since we did not have these individual values available to us. In the future, when our algorithm is deployed directly at each RNC, we will be able to measure the second moment as well to estimate the variance of the RTT (or other) values. Since it was infeasible to measure estimates for all possible combinations of tuples, we instead sampled one hundred tuples at random and computed the average relative error for these combinations. The variance of these measurements was found to be small.

6.3 Results for Real Data

We tested our algorithms on the cellular traffic data described above. For anomaly detection and other change detection problems, it suffices to have a relative estimation error that is within 15% since we are only interested in detecting significant changes from the normal values. For instance, we may not want to sound an alarm until the RTT has nearly doubled from its usual value. Of course, in other applications we may want to bound the error to under a few percent; our evaluation extends to this range as well.

Unlike the experiments on synthetic data, we did not have the ability to change all parameters (e.g., relative set size or intersection ratio). Also note that it was infeasible to compute M_2 for the real data because they had already been aggregated into flow records. Therefore, we focused on M_0 and M_1, which can be computed from the aggregated data, and the average ($Avg = M_1/M_0$).

We first investigated what parameters work well for our sketch on the real data. In Figure 9 we show the result when the number of buckets and the bucket size are simultaneously



(a) 0.01 i.r. (b) 0.02 i.r.

(c) 0.05 i.r. (d) 0.10 i.r.

Figure 10: Results of mean intersection relative errors when varying memory (buckets) for real data. Bucket size set to 16

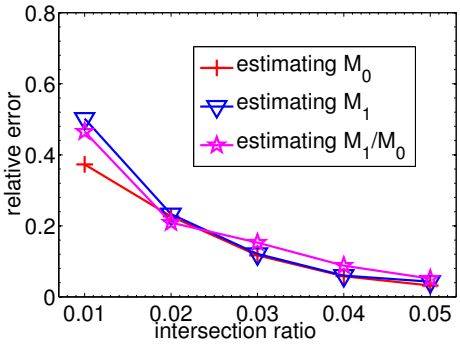


Figure 12: Results of mean intersection relative errors when varying intersection ratio for real data. Bucket size set to 16 and buckets to 32

varied, keeping the total number of counters fixed to 4096. As before, while keeping total memory usage fixed there is no significant change when varying the number of buckets and the bucket size, though we found that the results are slightly better if we make the bucket size smaller and number of buckets bigger. Hence, just as in the synthetic simulations, we chose to make the number of buckets as large as possible. For the remaining experiments we set the bucket size to 16.

In Fig. 11 we plotted the effect of varying the number of buckets on the relative error of the estimate when fixing the bucket size to 16. In this case there was a clear trend in the relative error of the measurement. From the plot, we can see that using memory of just $1024 \times 16 = 16384$ (buckets number $\times$ bucket size) is sufficient to retain high fidelity in the sketched data—under 5-10% relative error. We also found that using $256 \times 16 = 4096$ counters is sufficient to keep the error small enough for anomaly detection. Note that these results will only get better as the data sets available to us grow in size since the size of the data will not greatly affect the accuracy of the sketch. In particular, we foresee being able to maintain sketches of similar size even for packet-level data traces of several terabytes.

The results in Fig. 11, while satisfying, did not quite match up to those in our synthetic data. On closer examination, it turned out that there were several pairs for which the intersection ratio was considerably smaller than the 0.01-0.1 regime that we have been studying and are interested in studying. Recall that if the intersection ratio is very small (e.g., 0.001 or less), then it is not of significant interest in our application since it indicates a negligible-sized pairing. To focus our attention on the more interesting pairs, we re-ran the experiments sampling pairs from ones that had intersection ratio in the desired range. The corresponding plots for intersection ratios of at least 0.01, 0.02, 0.05 and 0.10 are shown in Fig. 10. For these data, we see that there is a considerable drop in the average relative error. In fact, for intersection ratio of 0.10 we see that even with $16 * 16 = 256$ counters we are able to reduce the relative error to well under 10%. As expected, the performance of the algorithm is considerably improved when the intersection ratio is higher.

Finally, to visualize the effect of intersection ratio on the accuracy of the sketch, we fixed the bucket size to 16 and measured the average relative error when sampling pairs that have at least some minimum intersection ratio. This is shown in Fig. 12. Once again, we see a substantial drop in the relative error within the 0.01 to 0.05 intersection ratio range

that we are aiming for. This indicates that we can use considerably smaller sketch sizes (e.g., in the 10-100 KB range) if we are interested only in the pairs that have significant intersection ratio.

In summary, we were able to run our algorithm on data collected by a major cellular carrier over a one-month period and shrink the memory footprint by more than two orders of magnitude while introducing under ten percent error on the estimate of the average RTT value. We found that if the intersection ratio is high, as is expected in most of the target applications, we can use even smaller sketches or achieve smaller error, or both.

7. RELATED WORK

Our work falls at the intersection of data streaming algorithms, cellular network performance, and association rule mining.

Data streaming algorithms. The seminal work of Alon et al. [2] for estimating the frequency moments of a stream opened up the research field of data streaming algorithms. In particular, the Tug-of-War sketch, which originated in [2] and was given that name in [1], is a simple but elegant tool for estimating M_2 of a stream that we take advantage of in this paper. Other work on mining data streams used a variety of techniques. For instance, Lakhina et al. [14, 12, 13] studied the diagnosis and characterization of feature distributions of network-wide anomalies in streams. Liu et al. [15] proposed PCA- and sketch-based streaming algorithms for network traffic anomaly detection. Yang et al. [25] studied the computational partitioning method between mobile devices and the cloud to achieve optimal speeds for processing streaming data.

Cellular network performance. Various techniques have been studied in recent years for measuring and analyzing cellular network metrics and performance. Shafiq et al. proposed characterizing M2M traffic patterns in cellular networks and studied that RTT measurements for TCP flows in [19]. They also analyzed Internet traffic dynamics of cellular network devices in [20]. Wang et al. implemented a tool that can unveil carriers' NAT and firewall policies by conducting intelligent measurement and thus help inform developers optimizing mobile applications and network configurations [23]. They also characterized the geospatial dynamics of application usage in 3G cellular networks in [18] and studied cellular network issues in [17] by quantifying aggregate network load and characterizing user-level traffic sessions. Falaki et al. [7] studied the detailed components of smartphone traffic and packet loss. Trestian et al. [22] showed how the relationship of mobile network users to their locations can benefit cellular network providers and location-based services. Balasubramanian et al. [3] provided a measurement study of the three types of mobile network technologies: 3G, GSM, and WiFi. Xu et al. [24] focused on identifying diverse usage patterns of smartphone apps in spatial, temporal, user, and device dimensions in cellular networks. Erman et al. [6] studied caching techniques for video stream traffic generated by smartphones in cellular network. Gember et al. [8] studied in-context network performance when users interact with their mobile devices.

Association Rule Mining. There is much literature on the topic of association rule mining; for example, see [5, 9] and references therein. Jin and Agrawal [11] in particular study ARM in the context of streaming data. ARM techniques have also been used for network troubleshooting. For example, Qiu et al. [16] used standard ARM techniques to mine router syslogs for network performance anomalies. Brauckhoff et al. [4] used ARM for detecting anomalies in backbone networks. As noted earlier, rather than proposing new ARM algorithms, this work reduces the amount of storage needed by an existing ARM algorithm for processing a massive data stream.

8. CONCLUSIONS

Change and anomaly detection are tasks essential for all cellular network providers these days. In this paper, we tackle this problem by proposing a novel scheme, which we call Crossroads, to find the anomalous events such as longer-than-usual RTT in cellular network data. To do this, we introduced the technique of intersecting pairs of network data stream digests of overlapping streams. In particular, we provide intersection algorithms for estimating the M_2, M_1, and M_0 of values in a data stream, which allow us to reduce the storage cost of this diagnostic data drastically, from $O(n)$ to $O(\sqrt{n})$. Our evaluations on synthetic and real-world data from a major cellular service carrier in the US generate very accurate and rapid estimates for detecting anomalies or other changes, demonstrating that our algorithms are quite reliable in practice.

9. REFERENCES

[1] Alon, N., Gibbons, P. B., Matias, Y., and Szegedy, M. Tracking join and self-join sizes in limited storage. In *Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems* (1999), ACM, pp. 10–20.

[2] Alon, N., Matias, Y., and Szegedy, M. The space complexity of approximating the frequency moments. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing* (1996), ACM, pp. 20–29.

[3] Balasubramanian, N., Balasubramanian, A., and Venkataramani, A. Energy consumption in mobile phones: a measurement study and implications for network applications. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference* (2009), ACM, pp. 280–293.

[4] Brauckhoff, D., Dimitropoulos, X., Wagner, A., and Salamatian, K. Anomaly extraction in backbone networks using association rules. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference* (New York, NY, USA, 2009), IMC '09, ACM, pp. 28–34.

[5] Cheng, J., Ke, Y., and Ng, W. A survey on algorithms for mining frequent itemsets over data streams. *Knowledge and Information Systems 16*, 1 (2008), 1–27.

[6] Erman, J., Gerber, A., Ramadrishnan, K., Sen, S., and Spatscheck, O. Over the top video: the gorilla in cellular networks. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference* (2011), ACM, pp. 127–136.

[7] Falaki, H., Lymberopoulos, D., Mahajan, R., Kandula, S., and Estrin, D. A first look at traffic on smartphones. In *Proceedings of the 10th ACM*

SIGCOMM conference on Internet measurement (2010), ACM, pp. 281–287.

[8] Gember, A., Akella, A., Pang, J., Varshavsky, A., and Caceres, R. Obtaining in-context measurements of cellular network performance. In *Proceedings of the 2012 ACM conference on Internet measurement conference* (2012), ACM, pp. 287–300.

[9] Han, J., Cheng, H., Xin, D., and Yan, X. Frequent pattern mining: Current status and future directions. *Data Min. Knowl. Discov. 15*, 1 (Aug. 2007), 55–86.

[10] Indyk, P. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on* (2000), IEEE, pp. 189–197.

[11] Jin, R., and Agrawal, G. An algorithm for in-core frequent itemset mining on streaming data. In *Proceedings of the Fifth IEEE International Conference on Data Mining* (Washington, DC, USA, 2005), ICDM '05, IEEE Computer Society, pp. 210–217.

[12] Lakhina, A., Crovella, M., and Diot, C. Characterization of network-wide anomalies in traffic flows. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement* (2004), ACM, pp. 201–206.

[13] Lakhina, A., Crovella, M., and Diot, C. Diagnosing network-wide traffic anomalies. In *ACM SIGCOMM Computer Communication Review* (2004), vol. 34, ACM, pp. 219–230.

[14] Lakhina, A., Crovella, M., and Diot, C. Mining anomalies using traffic feature distributions. In *ACM SIGCOMM Computer Communication Review* (2005), vol. 35, ACM, pp. 217–228.

[15] Liu, Y., Zhang, L., and Guan, Y. Sketch-based streaming pca algorithm for network-wide traffic anomaly detection. In *Distributed Computing Systems (ICDCS), 2010 IEEE 30th International Conference on* (2010), IEEE, pp. 807–816.

[16] Qiu, T., Ge, Z., Pei, D., Wang, J., and Xu, J. What happened in my network: Mining network events from router syslogs. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement* (New York, NY, USA, 2010), IMC '10, ACM, pp. 472–484.

[17] Shafiq, M. Z., Ji, L., Liu, A. X., Pang, J., Venkataraman, S., and Wang, J. A first look at cellular network performance during crowded events. In *Proceedings of the ACM SIGMETRICS/international conference on Measurement and modeling of computer systems* (2013), ACM, pp. 17–28.

[18] Shafiq, M. Z., Ji, L., Liu, A. X., Pang, J., and Wang, J. Characterizing geospatial dynamics of application usage in a 3g cellular data network. In *INFOCOM, 2012 Proceedings IEEE* (2012), IEEE, pp. 1341–1349.

[19] Shafiq, M. Z., Ji, L., Liu, A. X., Pang, J., and Wang, J. A first look at cellular machine-to-machine traffic: large scale measurement and characterization. In *ACM SIGMETRICS Performance Evaluation Review* (2012), vol. 40, ACM, pp. 65–76.

[20] Shafiq, M. Z., Ji, L., Liu, A. X., and Wang, J. Characterizing and modeling internet traffic dynamics of cellular devices. In *Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems* (2011), ACM, pp. 305–316.

[21] Snoeren, A. C., Partridge, C., Sanchez, L. A., Jones, C. E., Tchakountio, F., Kent, S. T., and Strayer, W. T. Hash-based ip traceback. In *ACM SIGCOMM Computer Communication Review* (2001), vol. 31, ACM, pp. 3–14.

[22] Trestian, I., Ranjan, S., Kuzmanovic, A., and Nucci, A. Measuring serendipity: connecting people, locations and interests in a mobile 3g network. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference* (2009), ACM, pp. 267–279.

[23] Wang, Z., Qian, Z., Xu, Q., Mao, Z., and Zhang, M. An untold story of middleboxes in cellular networks. *ACM SIGCOMM Computer Communication Review 41*, 4 (2011), 374–385.

[24] Xu, Q., Erman, J., Gerber, A., Mao, Z., Pang, J., and Venkataraman, S. Identifying diverse usage behaviors of smartphone apps. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference* (2011), ACM, pp. 329–344.

[25] Yang, L., Cao, J., Yuan, Y., Li, T., Han, A., and Chan, A. A framework for partitioning and execution of data stream applications in mobile cloud computing. *ACM SIGMETRICS Performance Evaluation Review 40*, 4 (2013), 23–32.

[26] Zhao, H. C., Lall, A., Ogihara, M., Spatscheck, O., Wang, J., and Xu, J. A data streaming algorithm for estimating entropies of od flows. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement* (2007), ACM, pp. 279–290.