"Estimating the number of differences between remote sets"

S. Agarwal and A. Trachtenberg

CISE Technical Report

CISE Technical Report #2004-IR-0001

CISE Technical Report Date: June 28, 2004

Estimating the number of differences between remote sets

Sachin Agarwal Ari Trachtenberg

Abstract

We propose several protocols for estimating the number of differences between sets held on remote hosts. Our approaches are based on modifications of the counting Bloom filter and are designed for minimizing communication and interaction between the hosts. As such, they are appropriate for streamlining a variety of communication sensitive network applications, including data synchronization in mobile networks, gossip protocols and content delivery networks. We provide analytical bounds on the expected performance of our protocols, together with heuristic improvements. We also provide experimental evidence that our protocols can outperform existing difference estimation techniques, and we demonstrate the value of this improvement in two sample network applications.

I. Introduction

Many network applications and protocols distribute identical databases over many hosts. Such distribution affords hosts parallel access and redundant backup of data. In the case of mobile networks, this distribution also permits access to the data for intermittently networked hosts. Personal Digital Assistants (PDAs) provide a prime example of such access, intermittently connecting to nearby desktop computers, laptops, other PDAs, or wireless networks according to availability. In order to maintain even weak data consistency, hosts must periodically reconcile their differences with other hosts as connections become available or according to prescribed scheduling.

Unfortunately, there is always a cost associated with reconciliation [1] such as network bandwidth utilization, latency of synchronization, battery usage (in battery-operated devices) and lost up-time of the database while the synchronization procedure completes. This cost may become significant in a multi-host network or if resources are severely constrained. Given the emergence of dense ad-hoc and sensor networks, a host could find itself reconciling very often with its peers in order to maintain minimal data consistency guarantees. However, in many cases a full-blown reconciliation is unnecessary, such as if two hosts are holding fairly similar content. It is the ability to discern such a condition that we address in this paper.

In a dense or constrained network the decision to reconcile should be based, in part, on the number of differences between the reconciling hosts' data sets. Although hosts with many differences between them should probably be fully reconciled, hosts that are fairly similar might wait for more differences to accumulate. Unfortunately, simple solutions, such as providing timestamps for updates on each host, do not scale well to dynamic or large networks because of the need to maintain an update history with respect to every other host [1, 2].

In this paper we introduce new approaches for estimating the number of differences between two data sets based on variants of counting Bloom filters. Formally, the problem is as follows: given two hosts A and B with data sets S_A and S_B respectively, we wish to estimate the size of mutual difference $S_A \oplus S_B \hat{=} (S_A - S_B) \cup (S_B - S_A)$. Our goal is to measure this size as accurately as possible and using as little communication as possible, measured both in terms of transmitted bytes and rounds of communication. As a secondary goal, we also seek to reduce the computational cost involved with such an estimation.

The authors are with the Department of Electrical and Computer Engineering at Boston University, Boston, MA. This work is based, in part, upon work supported by the National Science Foundation under Grants CCR-0133521 and ANI-0240333.

The ability to estimate the number of differences between two data sets inexpensively is of fundamental importance to networking applications that maintain weakly consistent data in the face of constraints on power, communication, or computation. We describe two examples of the utility of such an estimate in this paper: data synchronization and gossip protocols. In the first case, a difference estimate can be used for choosing an appropriate data synchronization protocol to minimize communication constraints [3–6]. In the second case, the estimate can be used to fine-tune gossiping applications [7] or to manage overhead in data replication services such as CODA [8], BAYOU [9], or SyncML [10].

A. Organization

In Section II we provide a baseline information-theoretic analysis of the difference estimation problem, giving lower bounds and inapproximability results. Thereafter, we describe some existing protocols for difference estimation and very briefly review traditional Bloom filters. In Section III we describe and optimize a difference estimation technique based on Bloom filters. In Section IV we introduce an alternative *wrapped filter* technique for estimating differences based on the counting Bloom Filter. The accuracy of this technique depends on the amount of probabilistic false positives incurred, and we discuss how to heuristically mitigate these in Section V-D. Finally, in Section VI we experimentally compare our approach to existing estimation techniques and demonstrate how the use of our approach yields significant performance improvement in two sample networking applications. Conclusions and directions for future work are discussed in Section VII.

II. Preliminaries

In this section we begin by providing an information-theoretic analysis of the amount of communication needed for difference estimation, and thereafter proceed to describe existing solutions to the problem and some fundamental properties of the Bloom filter.

A. Information-theoretic bounds

All the techniques and algorithms discussed in this paper compute the *approximate* number of differences between sets on remote hosts. Unfortunately, determining the *exact* number of set differences requires a large amount of communication. Specifically, such an exact determination requires communication that is at least linear in the size of one of the sets; in other words, one cannot do better (in terms of communication complexity) than simply transferring one of the sets to the other host for a local comparison.

Lemma II.1 *The number of differences between remote sets S, S', which are subsets of a common universal set $\mathcal{U}$, cannot be determined with less than $|\mathcal{U}| - 2$ bits of communication.*

Proof: Yao [11] showed that the minimum communication needed for two remote hosts to interactively compute a boolean function f on $M \times N$ is given by $\log_2(d(f)) - 2$ bits, where $d(f)$ is the minimum number of monochromatic rectangles needed to partition f on $M \times N$. As such, one can see that determining whether $S, S' \subseteq \mathcal{U}$ are equal (*i.e.,* interactively computing the identity function on $2^{\mathcal{U}} \times 2^{\mathcal{U}}$) requires at least $|\mathcal{U}| - 2$ bits of communication. Computing whether two sets are equal is clearly a special case of determining the number of differences between them, hence the bound. ■

The same communication complexity applies to an algorithm that approximates the number of differences between two sets within a multiplicative constant, since such an algorithm would determine set equality as a special case. The following lemma shows that approximating differences within an *additive* constant requires a similarly high communication complexity.

Lemma II.2 *Consider an algorithm computing an estimate $A(S, S')$ of the number of differences $\Delta(S, S')$ between two sets S, S'. If A returns an estimate within k of the actual number of differences* i.e.,

$$\Delta(S, S') - k \leq A(S, S') \leq \Delta(S, S') + k \quad \forall S, S' \subseteq \mathcal{U}$$

then A must communicate $\Omega(\mathcal{U})$ bits.

Proof: Consider the boolean function $f(S, S')$ defined to be 1 exactly when $A(S, S') \leq k$. Clearly computing A requires at least as much communication as computing f. On the other hand, the number of ones in any row $f(S, S') \ \forall \ S' \subseteq \mathcal{U}$ will, at most, consist of all sets that differ by $\pm k$ elements from S; there are $O(|\mathcal{U}|^{2k})$ such sets. As such $\frac{2^{|\mathcal{U}|}}{|\mathcal{U}|^{2k}}$ monochromatic rectangles are needed to partition the space of f, giving a minimum communication complexity of $|\mathcal{U}| - 2k \log(|\mathcal{U}|) - 2 \in \Omega(|\mathcal{U}|)$ bits. In fact the result can be generalized to any approximation that results in a function f with asymptotically less than $2^{|\mathcal{U}|}$ ones in any row. ■

As a result, it is clear that any protocol that correctly estimates set differences within any multiplicative or additive constant must effectively transmit the entirety of one of the sets. We conjecture that the same is true (asymptotically) for approximations within linear functions of the actual number of differences. In effect, it appears that one may not efficiently determine set differences with deterministic precision. However, as is often the case, tolerance to small amounts of error can significantly improve communication complexity.

B. Existing solutions

Various existing techniques for estimating set difference size are surveyed nicely in [5]. We present these and other currently used techniques next.

A straightforward approach to determining the number of differences between two hosts is to have one host transmit its entire set to the other host for a local count of the number of differences. Though this method will always provide an exact estimate of the number of differences between between the two host sets, it will do so at the expense of a large communication complexity, linear in the size of one host set.

One simple protocol for such an estimate involves random sampling, in which host A transmits k randomly chosen elements to host B for comparison. If B has r of the transmitted elements, then we can estimate that $\frac{r}{k}$ of the elements of B are common to A. The main problems with random sampling are a high error rate and low resolution, as we shall see in Section VI; in effect, a large number of samples (and hence high communication complexity) are required for reasonable accuracy.

Another approach to difference estimation involves the use of Bloom filters [12, 13] as we describe in detail in Section III. The inaccuracy of this estimate comes from false positive fits of data into the Bloom filter, but this inaccuracy can be made very small at the expense of high communication complexity. Our approach in this work extends the Bloom filter estimation scheme to improve both accuracy and communication.

One may also use a set reconciliation scheme such as CPISync [6, 14] to fully synchronize the two host sets and, in the process, determine the number of differences between them. Though the communication complexity of this scheme will be linearly dependent on the number of differences between the two sets, the multiplicative constants hidden by this linear dependence, and the number of rounds of communication, can be higher than necessary.

The problem of determining similarity in documents is also clearly related to our work in this paper, though its solutions are generally more complicated due to the relative complexity of the similarity metric. Determining document similarity is useful in designing search engines that discern and index similar documents on the Web [15, 16]. These approaches are based on clever sampling based techniques called min-wise sketches. Most of these approaches work by dividing the document into smaller units of information and then determining the percentage of similarity between these sets. Both random sampling and min-wise sketches have random data or hashes that are transmitted from one host to the other and these transmissions do not lend themselves to good compression in general. Bloom filters and their variants however are very amenable to high data compression, thus leading to small transmit sizes [17].

Similar approaches are also suggested for finding similarities across data and files in a large file system [18]. Traditional string distance metrics like Hamming and Levenshtein [19, 20] are not useful in these contexts because they require a pair-wise comparison of documents that is infeasible when there are a large number of documents.

C. Bloom filter basics

Bloom filters [12, 13, 17] are used to perform efficient membership queries on sets. The Bloom filter of a set is simply a bit array; each element of the set is hashed with several hashes into corresponding locations in the array, which are thereby set to 1 and otherwise 0. To test whether a specific element x is in a set, one need only check whether the appropriate bits are 1 in the Bloom filter of the set; if they are not, then x is certainly not in the set, but otherwise the Bloom filter reports that x is in the set. In the latter case, it is possible for the Bloom filter to incorrectly report that x is an element of the set (*i.e.,* a false positive indication) when, in fact, it is not.

The probability of a false positive of a Bloom filter for a set S is denoted $P_f(S)$ and depends on the number of elements in the set $|S|$, the length of the Bloom filter m, and the number of (independent) hash functions k used to compute the Bloom filter. This false positive probability is given in [13]) as

$$P_f(S) = \left(1 - \left(1 - \frac{1}{m}\right)^{k|S|}\right)^k . \tag{1}$$

III. Efficient synchronization of Bloom filters

One may use Bloom filters to estimate the number of differences between two sets. Specifically, host A (with set S_A) can compute the Bloom filter $\mathcal{B}_{S_A}$ of its data and transmit this filter together with $|S_A|$ to host B. By investigating which of its elements *fit* into $\mathcal{B}_{S_A}$ [1], host B can estimate the intersection size $|S_A \cap S_B|$ and thereafter the number of mutual differences between the two sets:

$$|S_A \oplus S_B| = |S_A| + |S_B| - 2|S_A \cap S_B|. \tag{2}$$

Lemma III.1 *When estimating the set difference of sets S_A and S_B using a k-hash Bloom filter of size m, the resulting estimate for host B will, in expectation, be low by*

$$2(|S_B - S_A|)P_f(S_A)$$

elements.

Proof: Assuming no transmission errors, any of the elements in $S_B - S_A$ will appear as a false positive in A's Bloom filter with probability given by (1). Each such false positive will reduce our estimate for the number of differences by two elements, as in (2). The statement then follows by linearity of expectation over probes from $S_B - S_A$. ■

Lemma III.1 also follows, without loss of generality, with A and B interchanged. Further, since $|S_B - S_A| \leq |S_B \oplus S_A|$, we may deduce that a Bloom filter estimate $\hat{\Delta}$ is related to the true set difference $\Delta = |S_A \oplus S_B|$ as follows:

$$(1 - 2P_f(S_A))\,\Delta \quad \leq \hat{\Delta} \leq \quad \Delta \tag{3}$$

[1] An element s is said to fit into a Bloom filter $\mathcal{B}$ if $\mathcal{B}(h_i(s)) = 1$ for all associated hash functions h_i.

The left-hand coefficient of Δ in (3) thus denotes the percentage error that we can expect from our estimate. The derivative of this coefficient with respect to m is always positive for positive integers m, k, and $|S_A|$, resulting in the following corollary.

Corollary III.2 *The Bloom filter difference estimate computed by host B will be within an expected ϵ fraction of the correct value if*

$$m \geq \frac{1}{1 - \left[1 - \left(\frac{\epsilon}{2}\right)^{\frac{1}{k}}\right]^{\frac{1}{k|S_A|}}}.$$

¿From the perspective of communication, it is simplest for one host to send over its entire Bloom filter, bit by bit, to the other host for the purposes of set difference estimation. However, in certain cases there may be a more efficient solution. The key observation in this regard is that Bloom filters on two remote hosts can be substantially similar if hosts' sets are very similar or very large (with respect to the size of the Bloom filter). As such, it might be more efficient to *synchronize* two Bloom filters using an efficient protocol such as CPISync [6, 14], rather than transmitting the Bloom filters in their entirety. This is because the CPISync algorithm and its various extensions [4] synchronize two data sets (with high probability) with communication that is proportional to the number of differences between the sets. The main deficit of this technique is that it is no longer non-interactive, and thus typically requires a number of rounds communication. The following theorem provides an analytical approach to deciding whether to transfer a Bloom filter in its entirety or to use a data synchronization protocol.

Theorem III.3 *The number of differences between two k-hash Bloom filters of length m of sets S_A and S_B is given by*

$$m\left[\left(1-\frac{1}{m}\right)^{|S_A|k} + \left(1-\frac{1}{m}\right)^{|S_B|k} -2\left(1-\frac{1}{m}\right)^{(|S_A \cup S_B|)k}\right] \xrightarrow{m \longrightarrow \infty} |S_A \oplus S_B|\, k. \qquad (4)$$

Proof: Denote by $w(i-1)$ the Hamming weight (*i.e.*, the number of 1's) of a random Bloom filter after $i-1$ insertions. An independently hashed new entry will collide with a 1 already in the filter with probability $1 - \frac{w(i-1)}{m}$, giving rise to the following recurrence, by linearity of expectation:

$$E[w(i)] = E[w(i-1)] + \left(1 - \frac{E[w(i-1)]}{m}\right).$$

Solving this recurrence with constraints $w(0) = 0$ and $w(1) = 1$ [21] yields:

$$E[w(i)] = m\left(1 - \left(1 - \frac{1}{m}\right)^i\right).$$

To derive the result, note that, if $\delta = |S_A \cap S_B|$, then there are $w(\delta k)$ ones common to both Bloom filters $\mathcal{B}_{S_A}$ and $\mathcal{B}_{S_B}$ due solely to elements common to S_A and S_B. Beyond these common elements, there are $w_1 = w(|S_A|k) - w(\delta k)$ additional ones in $\mathcal{B}_{S_A}$ and $w_2 = w(|S_B|k) - w(\delta k)$ ones in $\mathcal{B}_{S_B}$. However, some of these additional ones are common to both Bloom filters, due to hash collisions. Excepting these common ones, we are left with a Hamming weight $w_3 = w((|S_A| + |S_B| - \delta)k) - w(\delta k)$, determined by considering inserting all non-common set elements in S_A and S_B into a new Bloom filter. As such, the number of differences between both Bloom filters is given by

$$w_3 - (w_1 + w_2 - w_3) = 2w_3 - w_1 - w_2,$$

which leads to the theorem statement after algebraic manipulations. ■

Thus, on the one hand, bit-by-bit Bloom filter transfer will require m bits of communication. On the other hand, CPISync requires communication roughly equal to the number of differences in (4) multiplied by the size of a data item, $\log(m)$. Thus, for sufficiently large m, CPISync should be used whenever

$$2e^{\frac{|S_A \cup S_B|k}{m}} - e^{\frac{|S_A|k}{m}} - e^{\frac{|S_B|k}{m}} \leq \frac{1}{\log m},$$

where $e \approx 2.71828$ is the base of the natural logarithm.

Example III.4 *Consider estimating the number of differences between two remote sets, each with* $100,000$ 128*-bit elements. Suppose that we are satisfied with an estimate that is within* 25% *of the true number of differences.*

Wholesale set transfer would require a transmission of size roughly 1.5 MB*, but would provide an exact number of differences. For Bloom filter-based estimation, the smallest transmission size for which Corollary III.2 guarantees the desired* 25% *accuracy occurs when* $k = 3$ *and* $m = 432,807$*. Bit-by-bit transmission of such a Bloom filter would involve a transmission size of roughly* 53 KB*. On the other hand, CPISync synchronization of such a Bloom filter will expect less transmission as long as there are fewer than* $7,800$ *differences between the reconciling sets. If there are* $1,000$ *differences between the sets, the CPISync approach requires only roughly* 7 KB *for communication.*

IV. Wrapped filters

Wrapped filters hold condensed set membership information with more precision than a traditional Bloom filter. The additional precision comes at the expense of higher communication costs, but, surprisingly, this expense is outweighed by the benefits of improved performance. As we show later, wrapped filters often provide a more accurate estimate of set difference per communicated bit than traditional Bloom filters.

In Section IV-A we describe how to *wrap* (*i.e.,* encode) a set into a wrapped filter; this wrapping is identical to the encoding of counting Bloom filters. What is novel about wrapped filters, and the origin of their name, is the *unwrapping* (*i.e.*, decoding) process, through which a remote host can estimate its differences with a local host. We describe this unwrapping procedure in Section IV-B. Thereafter, in Section V we provide an analysis of the performance of this data structure, and in Section VI we show that wrapped filters can provide better estimates than Bloom filters.

A. Wrapping

Wrapped filters are constructed in a fashion similar to counting Bloom filters [12, 13]. A wrapped filter W_S of a set $S = \{s_1, s_2, s_3, \ldots s_n\}$ is first initialized with all zeroes, and then set elements are added to the filter by incrementing locations in $W(S)$ corresponding to k independent hashes $h_i(\cdot)$ of these elements, as depicted in Figure 1. More precisely, we increment $W_S[h_j(s_i)]$ for each set element $s_i \in S$ and hash function h_j in order to construct the wrapped filter W_S.

The wrapped filter clearly generalizes the Bloom filter in that we may transform the former into the latter by treating all non-zero entries as ones. Host A can use this Bloom filter property of a wrapped function to determine $|S_A - S_B|$ by inspecting B's wrapped filter W_{S_B}; in other words, all elements of S_A that do not fit the Bloom filter can be considered to be in $S_A - S_B$. Conversely, the unwrapping algorithm in Section IV-B will allow us to estimate $|S_B - S_A|$ from the same wrapped filter, resulting in an overall estimate for the mutual difference $|S_A \oplus S_B|$.

Unlike Bloom filters, wrapped filters (and counting Bloom filters) also have the feature of incrementally handling both insertions and deletions. Thus, whereas a traditional Bloom filter for a set would have to be recomputed upon deletion of an element, one may simply decrement the corresponding hash locations for this element in the wrapped filter. The price for this feature is the size of the filter, since each entry can

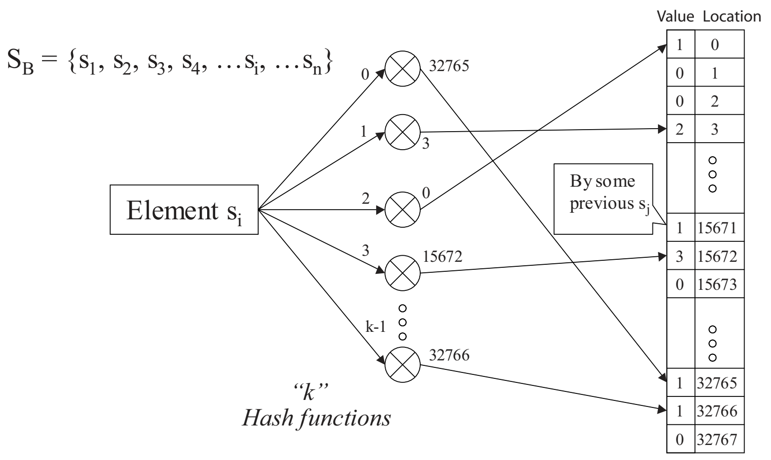


Fig. 1. Constructing an $m = 32,768$ long wrapped filter. We start with all zeros in the wrapped filter and for each $s \in S_B$ we calculate k separate hash functions and increment these k locations in the wrapped filter.

Protocol 1 Unwrapping a wrapped filter W_{S_B} against a host set S_A.

```
for each set element $s_i \in S_A$ do
    copy $W_{\text{temp}} = W_{S_B}$
    for each hash function $h_j$ do
        if $W_{\text{temp}}[(s_i)] > 0$ then
            $W_{\text{temp}}[h_j(s_i)] = W_{\text{temp}}[h_j(s_i)] - 1$
        else proceed to the next element $s_i$
    copy $W_{S_B} = W_{\text{temp}}$
return the estimate $\delta_A = \frac{\sum_{i=1}^{m} W_{S_B}[i]}{k}$
```

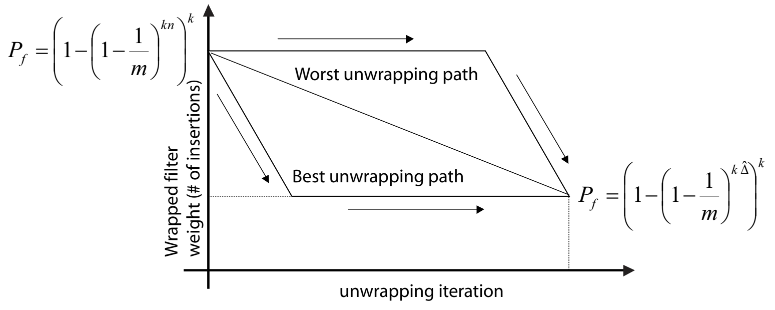
now take any of kn values (where $n = |S|$ is the size of the set being wrapped), requiring a worst-case of $m \log(kn)$ bits of storage memory and communication for a filter of size m; in contrast, traditional Bloom filters require only m bits of communication. Fortunately, the expected case is for each entry to have only $\frac{kn}{m}$ entries giving an expected multiplicative storage overhead of $\log(\frac{kn}{m})$ over traditional Bloom filters.

B. Unwrapping

We now describe how host A can unwrap a wrapped filter W_{S_B} to estimate $|S_B - S_A|$. This unwrapping procedure is presented formally in Protocol 1 but, at a higher level, it involves host A attempting to fit each of its set elements, one by one, into the wrapped filter. In this case a set element s is said to fit the wrapped filter if all k hash functions hash to non-zero locations in the filter. Once an element fits into the filter, all corresponding hash locations are decremented and A continues to attempt to fit subsequent set elements into the resulting filter. After all elements are compared against the filter, a final estimate of $\delta_A = |S_B - S_A|$ is calculated as the sum of the resulting filter entries divided by k.

The strength of the wrapped filter rests in two features of the unwrapping algorithm. First, the *total weight* of the wrapped filter (*i.e.*,$\sum_{s_i \in S} W_{S_B}(s_i)$) decreases as each set element is unwrapped. As a result, the false positive probability also generally decreases with each unwrapping, yielding a better overall estimate, as we shall see in Section V.

The second feature of the wrapped filter is that it can sometimes compensate for false positives. Intuitively, when a false positive element is unwrapped from the filter, it prevents (at least) one other non false positive



$P_f = \left(1-\left(1-\frac{1}{m}\right)^{kn}\right)^k$
Worst unwrapping path
Best unwrapping path
$P_f = \left(1-\left(1-\frac{1}{m}\right)^{k\hat{\Delta}}\right)^k$
Wrapped filter weight (# of insertions)
unwrapping iteration

Fig. 2. The wrapped filter weight never increases as the wrapped filter is unwrapped.

element from being unwrapped. Since we are only concerned with estimating the *number* of set differences, rather than actually determining the differences, this feature can mitigate the effect of the false positive.

V. Analysis and performance

A. False positive statistics

Like traditional Bloom filters, wrapped filters admit a small probability of false positives during the unwrapping process. A false positive occurs when the hash values $h_i(s)$ of an element s fit into the W_{S_B} despite the fact that $s \notin S_B$.

False positives can have two deleterious effects on our estimation procedure. First, they can reduce the overall set difference estimate by inappropriately reducing the weight of the wrapped filter. Secondly, a false positive can prevent a *valid* set element (*i.e.,* an element that is in the set intersection) from fitting in the resulting filter by reducing to zero (or causing to reduce to zero at some later time) one of the hash locations of the valid element. The inability to unwrap such valid elements results in unwanted residues at the conclusion of the unwrapping and thus adds error to our estimate.

¿From the perspective of false positives, a wrapped filter behaves like a changing Bloom filter.

Lemma V.1 *After i insertions into a wrapped filter, the probability of false positive is*

$$P_f(i) = \left(1 - \left(1 - \frac{1}{m}\right)^{ki}\right)^k .$$

Proof: The proof of the lemma follows analogously to the derivation of (1). After i insertions, the probability of a given location being zero is $\left(1 - \frac{1}{m}\right)^{ki}$. Thus, the probability of k locations being non-zero is precisely the statement of the lemma. ■

Alternatively, if a wrapped filter has total weight w (*i.e.,* the sum all its entries), then the probability of false positive is simply

$$P_f[w] = \left(1 - \left(1 - \frac{1}{m}\right)^{w}\right)^k , \tag{5}$$

because the independence of the hash functions allow us to view a total weight w wrapped filter as having been generated from a set of $\frac{w}{k}$ elements. In general, we shall denote by $w(i)$ the weight of a wrapped filter after i unwrapping iterations.

The most accurate estimates are achieved when elements in $S_A \cap S_B$ are unwrapped first, thereby reducing the false positive probability for other entries. The best and worst unwrapping paths are shown in Figure 2. The remaining cases can be summarized by a simple binomial-style upper bound. Specifically, the probability of ϕ false positives occurring during a random unwrapping by host A of W_{S_B} is *at most*

$$\binom{|S_A|}{\phi} \left(P_f(|S_A|)\right)^{\phi} \left(1 - P_f[\hat{\Delta}]\right)^{|S_A|-\phi} , \tag{6}$$

where $\hat{\Delta}$ is the estimate returned by the unwrapping algorithm.

B. Effects of false positives

False positives introduce error in our estimate of the number of differences between two sets. Since the weight of our wrapped filter decreases with each unwrapping iteration, so does the false positive probability (as per (5)). In fact, the following theorem shows that, in expectation, the decrease in weight of the wrapped filter will be linear in the number of unwrapping iterations. This results in the superior performance of wrapped filters over Bloom filters in some cases, as we shall confirm experimentally.

Theorem V.2 *Consider a k-hash wrapped filter W with initial false positive probability p_f being (uniformly) randomly unwrapped. If $d = |S_A - S_B|$ and $p_f \ll 1$ then the expected weight of W decreases by*

$$\left(k\frac{|S_A| - d}{|S_A|}\right)$$

at each unwrapping iteration.

Proof: Consider the i-th element $x_i \in S_A$ being unwrapped by host A from host B's wrapped filter W_{S_B}. There are two possibilities for x_i:

$x_i \notin S_B$ The element x_i might still fit into the wrapped filter as a false positive. For a correctly designed wrapped filter this probability should be very low.

$x_i \in S_B$ In this case x_i might either be correctly unwrapped from the filter, or it may fail to be unwrapped due to an earlier false positive(s) coinciding with one of its hashes. The probability of the latter case occurring, for any one hash function, is simply $\frac{z(i)}{m}$, where $z(i)$ is the number of zeroes introduced into W_{S_B} by the $f(i)$ false positives that have occurred up to the i-th iteration.

We next compute the expected decrease in weight of W_{S_B} with each iteration. Let Δ_i denote the decrease in the weight of the wrapped filter $W^i_{S_B}$ after the i-th element of S_A is unwrapped. Then,

$$\begin{aligned} E(\Delta_i) &= k \cdot \left\{\Pr\left(x_i \text{ fits } W^i_{S_B},\ x \notin S_B\right) + \Pr\left(x_i \text{ fits } W^i_{S_B},\ x \in S_B\right)\right\} \\ &= k \cdot \left\{P_f[w(i-1)] \cdot \frac{d}{|S_A|} + \Pr\left(x_i \text{ fits } W^i_{S_B} \mid x \in S_B\right) \cdot \frac{|S_A| - d}{|S_A|}.\right\} \end{aligned}$$

Since the the false positive probability is strictly non-increasing with unwrappings, and we have assumed that $P_f[w(0)] \ll 1$ (meaning that $P_f[W(i-1)]$ is small), we may conclude that

$$E(\Delta_i) \longrightarrow \Pr\left(x_i \text{ fits } W^i_{S_B} \mid x \in S_B\right) \cdot \frac{|S_A| - d}{|S_A|}. \tag{7}$$

For the one remaining unknown term, we refer to the beginning of the proof, noting that

$$\Pr\left(x_i \text{ fits } W^i_{S_B} \mid x \in S_B\right) = \left(1 - \frac{z(i)}{m}\right)^k. \tag{8}$$

For small probabilities of false positives, as assumed, the right hand side of (8) is very close to 1, thereby proving the theorem. ■

The significance of Theorem V.2 is that it identifies the expected wrapped filter behavior as corresponding to roughly the diagonal of the trapezium in Figure 2. This, in turn, determines (in expectation) the probability of error at any iteration of the decoding algorithm, and it is left as an open problem how to compute and correct for this bias.

We may also produce deterministic bounds on the effects of a false positive, as given by the following lemma.

Lemma V.3 *Each false positive will contribute an error of ε to our estimate, with*

$$-1 \leq \varepsilon \leq k-1 \tag{9}$$

Proof: A given false positive s can prevent at most k valid elements from being unwrapped, resulting in an increase of the wrapped filter weight by at most $k(k-1)$. At the other extreme, if all of the k positions decremented by unwrapping s were of invalid elements, then we incorrectly decrease the filter weight by k. ■

Note that for $k = 1$, the right hand size of (9) is 0, meaning that such wrapped filters will always produce a lower bound on the actual number of differences.

C. Wrapped filter size and compression

It is important that the size m of the wrapped filter be as small as possible so as to minimize communication complexity between two hosts that are estimating their set difference. However, it is also clear that the accuracy of the wrapped filter estimation relies on m being as large as possible, thereby reducing the probability of false positives. Fortunately, it is possible to compromise between these two requirements by using compression similarly to what has been done with Bloom filters [17].

We can compute the expected probabilities of a location being set to $i = 0, 1, 2, \ldots$ from the initial weight $w = n \cdot k$ of the wrapped filter and then use arithmetic coding [22] to come very close to the entropy bound while compressing these filters. The probability of a location being incremented to i while constructing the wrapped filter is given by

$$p_i = \binom{w}{i}\left(1-\frac{1}{m}\right)^{w-i}\left(\frac{1}{m}\right)^{i}.$$

This binomial distribution has mean $\frac{w}{m}$ and variance $w\frac{1}{m}(1-\frac{1}{m})$. The ratio $\frac{w}{m}$ is usually less than 1 when designing for small false positive values and thus the distribution of p_i's is narrow and the wrapped filter locations are populated by only a few distinct i's. Given these probabilities, the size of the wrapped filter is lower bounded by the entropy per location of the filter times the length of the filter:

$$\text{Filter size} \geq m \cdot \sum_i -p_i \cdot log(p_i) \tag{10}$$

We shall revisit the effects of such compression in Section VI.

D. Heuristics and refinements

The following heuristics promise to improve the wrapped filter, but we leave their complete analysis as a direction for future work.

a) Multiple decodings: The error in our estimate depends on the order in which a filter is unwrapped. It is thus better to first unwrap the common elements of S_A and S_B, thereby insuring that further false positives will only reduce the estimate by 1. Of course, we do not know *a priori* which elements are common to both sets; however we can unwrap the filter several times in different random orders so as to improve our estimate. Since no operation in the unwrapping algorithm is super linear, this approach is computationally feasible.

b) Unwrapping order: Wrapped filters work best when the elements in the set intersection are decoded before the other elements because this would have already reduced the false positive probability before we try to decode potential false-positives. There is no way of knowing which elements are in the intersection of the two sets beforehand but in many practical instances such as data-sets that have many additions and only a few deletions, the ‘older’ elements can be decoded before the recent additions. A good example of this is a address-book where contacts are more likely to be added rather than deleted.

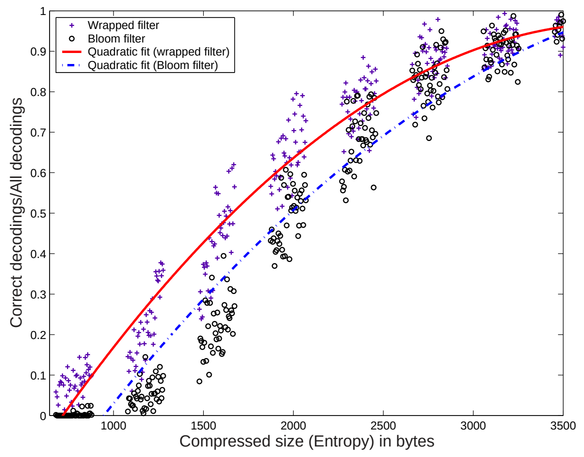


Fig. 3. Set difference computation using wrapped filters and Bloom filters. The probability of determining the exact number of differences is plotted versus the compressed size of the filters, which determines the communication complexity.

c) Guard Elements: As a last heuristic, we propose to introduce certain known (guard) elements in the sets. These guard elements are generated randomly according to a common random number generator. By counting the number of these guard elements that do not fit into the wrapped filter, we can appropriately adjust the estimate returned by the unwrapping algorithm.

E. *An example comparison*

Despite its larger size, the wrapped filter produces better set difference estimates than the standard Bloom filter for the same amount of communication. We illustrate this improvement with a simple but general example.

As our setting, consider two data-sets S_A and S_B held on hosts A and B that are intermittently connected. We assume that the sets are identical at time $t = 0$ and that each contains $1,000$ elements before the network connection is severed. We further assume that each host makes independent additions and deletions from its set and that the number of such modifications per unit time are Poisson distributed random variables with rates $\lambda_{Add} = 20$ and $\lambda_{Del} = 1$.

At time $t = 20$, a network connection is re-established between the two hosts, whereupon they each hold (in expectation) about $n = 1,400$ elements. The hosts now try to determine the exact number of set differences (*i.e.*, no tolerance for errors) between them using Bloom filters and wrapped filters.

In each case the use $k = ln(2) \cdot \frac{m}{|S|}$ hash functions, which produce the smallest initial false positive probability Bloom filters [17] and, straightforwardly, also in wrapped filters. We also assume that both filters are compressed nearly optimally using arithmetic coding, and that the wrapped filter decoding makes use of the known $1,000$ initial intersection of the two sets to improve decoding performance (as per the heuristic in Section V-D). Our results, in Figure 3 show that, for the same (compressed) communication complexity, the wrapped filter performs better than the Bloom filter. The 'clumping' of data-points in the figure results from discrete jumps in the computed value of the number of hash functions k.

VI. Experimental results

We now provide several experimental demonstrations of the efficacy and importance of our techniques. We compare this performance to various other estimation techniques in Section VI-A. Finally, in Section VI-B we show how efficient set estimation can significantly improve two sample networking applications.

In evaluating the effectiveness of wrapped filters, we used sets whose elements each contained 1K of data, comparable in size to an entry in a simple distributed address book or memo pad. For our hash functions, we

Transmission size (bytes)	Random	Minwise	Bloom	Wrapped
≈ 1000	126 ± 6.6705	99.467 ± 4.5121	93.2	99.2 ± 0.3578
≈ 3000	114.62 ± 1.7308	101.06 ± 1.3911	99.8 ± 0.8310	100 ± 0
≈ 6000	107.62 ± 1.2035	99.305 ± 0.91264	100 ± 0.4104	100 ± 0

Fig. 4. Performance of the wrapped filter, Bloom filter, min-wise sketches and, random sampling. 100 actual differences, averaged over 500 trials.

used the pseudo-random number generator (PRNG) from Victor Shoup's Number Theory C++ Library [23], seeded with a 128-bit MD-5 hash [24] of the element data; the MD-5 seeding insured consistent hash values for element data across different hosts. We found that the choice of the hash functions significantly affects overall performance.

Our experiments were averaged over 100 trials on an Intel Pentium-4 class machine and the time taken to construct and unwrap the wrapped filters was found to be insignificant.

A. Comparison of various techniques

Figure 4 depicts a comparison of random sampling, min-wise sketches, standard Bloom filter, and wrapped filter approaches in estimating the number of differences between two sets. The accuracy of the techniques is plotted as a function of their compressed communication complexity. As may be expected, the standard Bloom filter and wrapped filter approaches are very similar for very large filter sizes (relative to the size of the sets being compared).

Random sampling, on the other hand, is quite inaccurate unless almost all the samples are transmitted. min-wise sketches and random sampling have a high standard deviation as compared to Bloom filters and wrapped filters. As our analysis predicts, Bloom filters generally provide a lower (and less accurate) estimate of difference size than wrapped filters.

B. Applications

1) Data Synchronization - CPISync: The CPISync algorithm and its variants [4, 6, 14] allow two hosts to synchronize their set data efficiently with respect to both communication and computation complexity. The algorithm's performance can be significantly improved if a good estimate $\hat{\Delta}$ of the number of differences between reconciling data sets is known *a priori*. If this estimate is too high, then CPISync communicates unnecessarily many bits; on the other hand, if this estimate is too low, then CPISync requires more rounds and bits of communication than necessary. Thus, there is more of a penalty for low estimates than comparably high ones. In any case, the quality of the estimate does not improve the asymptotic worst-case performance of CPISync, only the multiplicative constants behind the asymptotic results [25].

For the purposes of our experiments, we have fed CPISync an initial estimate $\hat{\Delta}$ from the algorithms described in this work, namely Bloom filter and wrapped filter estimation each of size $m = 32,768$ and utilizing $k = 10$ hash functions. The reported communication complexity includes the cost of computing this initial estimate. Figure 5 compares the number of bytes transmitted for a synchronization of remote sets using the various types of initial estimates. We can see that the Bloom filter performance is especially poor because its estimate is *always* lower than the actual number of differences, thereby incurring a harsher communication penalty. We also compare the number of rounds of communication needed for the various set synchronizations, adding a half a round to account for filter transmission in the initial difference estimation stage.

2) Gossip Protocol: Gossip protocols [26, 27] spread information in a network through random exchanges of information between hosts. Each host (node) in the network randomly selects a neighbor with a certain gossip probability and synchronizes information with this neighbor. If the average degree of the

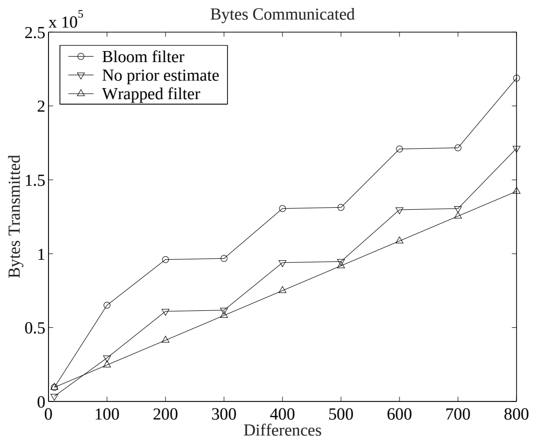


Fig. 5. CPISync total bytes used - using a Bloom filter and wrapped filter to estimate $\overline{\Delta}$. Each of the sets contains 1000 elements. All differences are symmetric.

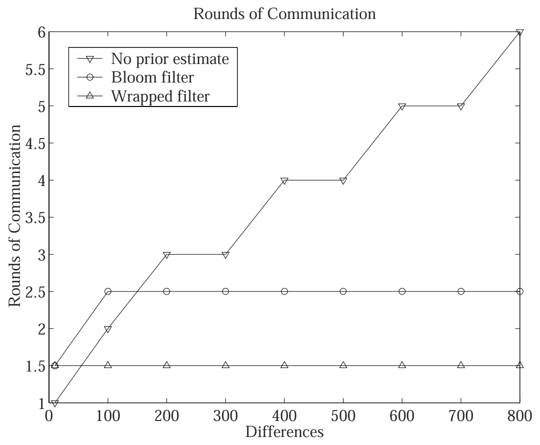


Fig. 6. CPISync total rounds of communication - using a Bloom filter and wrapped filter to estimate $\overline{\Delta}$. Each of the sets contains 1000 elements. All differences are symmetric.

network and the gossip probability are sufficiently high then it has been shown that the information will reach every node in the network with high probability.

Gossip protocols flood a network with a large number of messages, many of which are redundant because gossiping hosts may have little new information for each other. It is here that a prior estimate of the number of differences may reduce the overall communication, by letting hosts focus gossip on "interesting" neighbors (*i.e.*, those that differ by more than some threshold number of entries).

In our simulations, chosen neighbors agreed to gossip if they differed by at least 8K of data, or they had no gossips in the previous round (a degenerate condition). We used a Bloom filter and a wrapped filter of size $m = 1,024$ and $k = 6$ to estimate the number of differences between two hosts before they gossiped. Initially, each node had 4K (representing one set element) of new information to share with others and 396K of common information. Our network consisted of 100 nodes connected at random so as to have an average degree of 10. All results are averaged over 85 different network graphs.

Figure 7 shows, for different gossiping probabilities, the total number of bytes communicated until each host on the network learned all the new information available. The reported communication costs include both the gossiping and difference estimation costs. Interestingly, even through the overhead of one filter transmission is significantly lower for a Bloom filter than a wrapped filter, the Bloom filter's low estimates

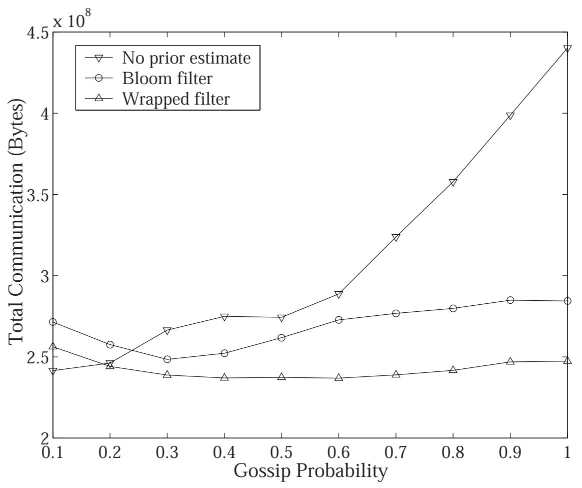


Fig. 7. Gossip protocol with and without prior difference estimation between gossiping hosts on a 100-node (connected) graph with average degree 10.

on differences cut off many useful gossips, resulting in a higher overall communication complexity.

Overall, the figure also demonstrates that having a prior estimate on the number of differences between gossiping hosts significantly improves the communication overhead of the system for high gossiping probabilities.

VII. Conclusions

We have analyzed and experimentally demonstrated several methods for estimating the number of differences between remote sets. We have also shown the impact of the communication efficiency of these techniques in two applications: data synchronization and gossip protocols.

The approaches we described are generally based on the Bloom filter. The first approach involved transmission of a Bloom filter from one host to another, either using wholesale data transfer or using a fast synchronization technique from the literature. Both transmission methods were found to be practical in different circumstances, and analytical means were developed for designing the Bloom filter to fit desired accuracy or communication requirements.

Our second approach involved a modification known as a wrapped filter, whose data structure is also known as a counting Bloom filter in the literature. Our novel decoding algorithm for the wrapped filter allows it to be generally more accurate than the standard Bloom filter for the same communication complexity. In addition, the wrapped filter technique is non-interactive and can be easily maintained in an incremental fashion as data is inserted or deleted from a set. The encoding and decoding process can also be made computationally efficient.

All these qualities make wrapped filters particularly suitable for the many network applications where there is a need to quickly measure the consistency of distributed information. Wrapped filter estimation is especially attractive when the sets being compared (i) have a common baseline at a none point in time (as in the example from Section V-E) or (ii) have relatively few differences, resulting in a quick reduction of filter weight during decoding.

Acknowledgments

The authors are grateful to John Byers for introducing us to the difference estimation problem, to Saikat Ray for stimulating discussions related to proper implementation of Bloom estimation, and to David Starobinski for stimulating discussions and feedback.

References

[1] S. Agarwal, D. Starobinski, and A. Trachtenberg, "On the scalability of data synchronization protocols for PDAs and mobile devices," *IEEE Network*, vol. 16, no. 4, July 2002.

[2] Michael Rabinovich, Narain H. Gehani, and Alex Kononov, "Scalable update propagation in epidemic replicated databases," in *Extending Database Technology*, 1996, pp. 207–222.

[3] "Palm developer on-line documentation," http://palmos/dev/tech/docs.

[4] Y. Minsky and A. Trachtenberg, "Scalable set reconciliation," in *Proc. 40-th Allerton Conference on Comm., Control, and Computing*, Monticello, IL., October 2002.

[5] J. Byers, J. Considine, M. Mitzenmacher, and S. Rost, "Informed content delivery across adaptive overlay networks," *ACM SIGCOMM*, August 2002.

[6] D. Starobinski, A. Trachtenberg, and S. Agarwal, "Efficient pda synchronization," *IEEE Trans. on Mobile Computing*, vol. 2, no. 1, January-March 2003.

[7] K. Guo, M. Hayden, R. van Renesse, W. Vogels, and K. P. Birman, "GSGC: An efficient gossip-style garbage collection scheme for scalable reliable multicast," Tech. Rep., Cornell University, December 1997.

[8] J. J. Kistler and M. Satyanarayanan, "Disconnected operation in the coda file system," *ACM Transactions on Computer Systems*, vol. 10, no. 1, pp. 3–25, 1992.

[9] D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser, "Managing update conflicts in bayou, a weakly connected replicated storage system," in *Proceedings of the 15th Symposium on Operating Systems Principles*, Copper Mountain Resort, Colorado, December 1995, ACM, number 22, pp. 172–183.

[10] "SyncML," http://www.syncml.org/.

[11] A. C. Yao, "Some complexity questions related to distributive computing," in *Proceedings of the 11th Annual ACM Symposium on Theory of Computing*, 1979, pp. 209–213.

[12] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, July 1970.

[13] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: a scalable wide-area Web cache sharing protocol," *IEEE/ACM Transactions on Networking*, vol. 8, no. 3, pp. 281–293, 2000.

[14] Y. Minsky, A. Trachtenberg, and R. Zippel, "Set reconciliation with nearly optimal communication complexity," *IEEE Trans. on Info. Theory*, September 2003, to appear.

[15] Broder, "On the resemblance and containment of documents," in *SEQS: Sequences '91*, 1998.

[16] Andrei Z. Broder, Moses Charikar, Alan M. Frieze, and Michael Mitzenmacher, "Min-wise independent permutations," *Journal of Computer and System Sciences*, vol. 60, no. 3, pp. 630–659, 2000.

[17] M. Mitzenmacher, "Compressed bloom filters," in *Proceedings of the Twentieth Annual ACM Symposium on Principles of Distributed Computing*, Newport, Rhode Island, USA, August 2001, pp. 144–150, ACM Press.

[18] U. Manber, "Finding similar files in a large file system," in *Proceedings of the USENIX Winter 1994 Technical Conference*, San Fransisco, CA, USA, 1994, pp. 1–10.

[19] D. S. Hirschberg, "Serial computations of Levenshtein distances," in *Pattern matching algorithms*, A. Apostolico and Z. Galil, Eds., pp. 123–141. Oxford University Press, 1997.

[20] R.A. Wagner and M.J. Fisher, "The string-to-String Correction Problem," *Journal of the ACM*, vol. 21, no. 1, pp. 168–173, January 1974.

[21] G. Leuker, "Some techniques for solving recurrences," *ACM Computing Surveys*, vol. 12, no. 4, pp. 419–436, 1980.

[22] J. Rissanen and G. G. Langdon, "Arithmetic coding," *IBM J. Res. Develop*, vol. 23, no. 2, pp. 149–162, March 1979.

[23] V. Shoup, "NTL: A library for doing number theory," http://shoup.net/ntl/.

[24] R.L. Rivest, "The md5 message digest algorithm," April 1992.

[25] A. Trachtenberg, D. Starobinski, and S. Agarwal, "Fast PDA synchronization using characteristic polynomial interpolation," *Proc. INFOCOM*, June 2002.

[26] A.J. Demers, D. H. Greene, C. Hause, W. Irish, and J. Larson, "Epidemic algorithms for replicated database maintenance," in *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, Vancouver, British Columbia, Canada, August 1987, ACM, number 6, pp. 1–12.

[27] Yaron M. Minsky, *Spreading Rumors Cheaply, Quickly, And Reliably*, Ph.D. thesis, Cornell University, Ithaca, NY, August 2002.