

Practical Set Reconciliation*

Yaron Minsky[†]
yminsky@cs.cornell.edu
Cornell University

Ari Trachtenberg[‡]
trachten@bu.edu
Boston University

February 26, 2002

Abstract

We consider the problem of efficiently reconciling two similar sets held by different hosts. This problem is motivated by the problem of data exchange in a gossip protocol, but also has other applications including synchronization of PDA databases and maintenance of routing tables in the face of host failures. Previous results have presented algorithms for set reconciliation that are nearly optimal in terms of communication complexity, but have computational complexity that is cubic in the number of differences. We present and analyze a novel algorithm that has expected computational and communication complexity that is linear in the number of differences between the sets being reconciled. We also provide experimental results from an implementation of this algorithm.

A version of this paper appeared as:

Y. Minsky and A. Trachtenberg, *Scalable Set Reconciliation*, 40th Annual Allerton Conference on Communication, Control, and Computing, Monticello, IL, October 2002.

*The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of these organizations or the U.S. Government.

[†]Supported in part by ARPA/RADC grant F30602-96-1-0317, AFOSR grant F49620-00-1-0198, Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory Air Force Material Command USAF under agreement number F30602-99-1-0533, National Science Foundation Grant 9703470, and a grant from Intel Corporation.

[‡]The work of Ari Trachtenberg is based upon work supported by the National Science Foundation under NSF Grant No. CCR-0133521

1 Introduction

The essential characteristic of a *gossip protocol* is that information exchanges, or *gossips*, occur between randomly selected pairs of hosts. Gossip protocols have proven an effective tool in the design of simple distributed protocols that are both efficient and robust [2–5, 9, 12, 15]. A fundamental issue in the design of a gossip protocol is determining what information hosts need to exchange in the course of a gossip. Demers *et al.* [2] proposed two approaches to this problem: *rumor-mongering* and *anti-entropy*. In rumor-mongering, hosts only exchange *hot* information that has been received recently and is therefore likely not to have been disseminated to all other hosts. In anti-entropy, each host stores a database with all the information it has obtained and fully reconciles this database with each host gossip partner.

Anti-entropy has been used successfully in a number of different gossip protocols [3, 5, 9, 12, 15]. The problem with anti-entropy, however, is that wholesale exchanging of databases is inefficient when there are few actual differences between the databases. This situation is, in fact, common in gossip protocols, particularly in situations where information is introduced into the system at a low rate.

Set reconciliation. The *set reconciliation* problem was proposed in [10, 11] in an attempt to improve the performance of anti-entropy. Consider a pair of hosts A and B , each holding a set S_A and S_B of b -bit bitstrings. The goal of set reconciliation is for A and B to each compute $S_A \cup S_B$ with a minimum of communication. Set reconciliation is connected to graph coloring and the construction of error-correcting codes [6, 7], and has a number of applications outside of gossip protocols. In particular, it has been applied to PDA synchronization and routing table maintenance [8, 13, 14]. More generally, set reconciliation can be used in a variety of systems where distributed information needs to be reconciled.

Earlier work in [10, 11] presented algorithms for solving set reconciliation with near-optimal communication complexity by interpolating rational functions over a finite field. These algorithms require sending an amount of data on the order of the size of the symmetric difference between S_A and S_B . Though very efficient from a communication perspective, these algorithms suffer from a computational complexity that is cubic in the size of the symmetric difference, rendering them impractical in many scenarios.

Our main contribution in this work is to provide a reconciliation algorithm whose expected computational and communication complexity are linear in the number of differences between reconciling hosts. Though the computational complexity of the algorithm is also necessarily linear in the size of the reconciling databases, we provide an incremental data structure that spreads this part of the computation over insertions and deletions to the database. This incrementality is especially important in applications such as gossip where hosts repeatedly reconcile the same evolving database.

Application to gossip protocols. The applicability of set reconciliation to gossip depends in part on how well the semantics of gossip reconciliation match up with the semantics of set reconciliation. There are two main issues here. The first is the fact that set reconciliation assumes that the data elements are fixed-length bitstrings, whereas gossip protocols often distribute variable-length data. This problem can be solved by simply running set reconciliation on fixed-length hashes of the data in question. The result of that reconciliation can then be used to determine what data each host is missing.

The second issue is that gossip protocols rarely reconcile their databases with a simple union.

Consider for example a gossip protocol where hosts distribute tuples of the form $\langle A, v \rangle$, where A is the name of the host that introduced the tuple, and v is a version number. In this example we assume that when two tuples have the same hostname, only the most recent version number is retained. This kind of reconciliation is used in [15], and similar types of reconciliation are common in anti-entropy based gossip protocols. Under this model, two sets

$$S_A = \{\langle A, 3 \rangle, \langle C, 7 \rangle, \langle D, 9 \rangle\} \quad S_B = \{\langle A, 2 \rangle, \langle B, 12 \rangle, \langle C, 9 \rangle, \langle D, 9 \rangle\}$$

would be reconciled as follows

$$\{\langle A, 3 \rangle, \langle B, 12 \rangle, \langle C, 9 \rangle, \langle D, 9 \rangle\}. \quad (1)$$

On the other hand, the union of the two databases is

$$\{\langle A, 2 \rangle, \langle A, 3 \rangle, \langle B, 12 \rangle, \langle C, 7 \rangle, \langle C, 9 \rangle, \langle D, 9 \rangle\}. \quad (2)$$

Nevertheless, (1) can be derived straightforwardly from (2) by dropping superceded tuples. Thus, even when the reconciliation required by a gossip protocol is not equivalent to set union, set reconciliation can be a useful intermediate step in reconciling the databases.

Outline. The rest of this paper is organized as follows. Section 2 provides an abstract description of the main algorithm presented in [11] and describes some of the problems associated with it. Section 3 presents our main result, which is an interactive algorithm based on repeated repartitioning of the universe of b -bit bitstrings that achieves linear expected computational complexity while maintaining linear expected communication complexity. Section 4 describes a data structure called a *partition-tree* which permits incremental maintenance of the data required by our interactive algorithm. Finally, Section 5 discusses our implementation and presents performance results.

2 Basic Set Reconciliation

The set reconciliation algorithms presented in [10, 11] are based on the use of a novel form of checksum that summarizes the contents of a set. The key property of this checksum is that it can be used to determine differences between sets. In particular, given only the checksums for two sets S_A and S_B , one can recover the symmetric difference $S_A \oplus S_B$, as long as $|S_A \oplus S_B|$ is not too large. Moreover, one can test with arbitrarily low probability of error whether the symmetric difference is too large. The checksum has two parameters:

- A *recovery bound* \overline{m} , such that reconciliation succeeds if the size m of the symmetric difference is less than or equal to \overline{m} .
- A *redundancy factor* k that determines the probability of detecting that reconciliation is not possible. In particular, if $m > \overline{m}$, that fact can be established with an error probability of ϵ_k , as defined below:

$$\left(\frac{|S_A| + |S_B|}{2^b} \right)^k, \quad (3)$$

The general framework for the algorithms in [10, 11] (collectively called BASIC-RECON) can be abstracted into four primitives, shown in Figure 1. The details of how these primitives are implemented can be found in [10, 11, 14]. The general idea is that the checksum of a set is computed

`init_cs(\overline{m} , k)`: Returns an initial checksum with recovery bound \overline{m} and redundancy factor k ; the size of the checksum is given in Equation (4).

`add_element(cs , β)`: Updates cs to reflect addition of β to host's set.

`del_element(cs , β)`: Updates cs to reflect removal of β from host's set.

`recover(cs_A , cs_B)`: If $|\Delta_A| + |\Delta_B| \leq \overline{m}$, then Δ_A and Δ_B are returned. Otherwise, fail is returned with probability given by (3).

Figure 1: BASIC-RECON primitives

	Computational Complexity
<code>init_cs(\overline{m}, k)</code>	$O(b(\overline{m} + k))$
<code>add_element(cs, β)</code>	$O(b(\overline{m} + k))$
<code>del_element(cs, β)</code>	$O(b(\overline{m} + k))$
<code>recover(cs_A, cs_B)</code>	$O(b\overline{m}^3 + b\overline{m}k)$

Table 1: Computational complexity of BASIC-RECON primitives.

as evaluations (over a finite field) of a polynomial derived from the set; the recovery process involves interpolation and factoring of a rational function whose zeroes and poles are elements of the symmetric difference of the two sets being reconciled.

Given a known bound \overline{m} on the size m of the symmetric difference, reconciliation can proceed as follows: Hosts A and B create and maintain checksums cs_A and cs_B corresponding to the sets S_A and S_B respectively. To initiate a reconciliation, host A sends cs_A to B . Host B then computes

$$(\Delta_A, \Delta_B) = \text{recover}(cs_A, cs_B),$$

and sends Δ_B to A . If no bound \overline{m} is known, then the protocol is executed with successively larger values of \overline{m} , until `recover` succeeds.

The size of a checksum with recovery bound \overline{m} and redundancy factor k is bounded by

$$(\overline{m} + k + 1)(b + 1) - 1 \tag{4}$$

bits. Note that when k is small, the size of the checksum is close to $\overline{m}b$, which is the size of the largest symmetric difference that can be reconciled with that checksum.

The computational complexity of the basic primitives is shown in Table 1. The main computational bottleneck in BASIC-RECON is the `recover` primitive, whose computation complexity is cubic in \overline{m} . The primitives `add_element` and `del_element` can also be problematic, particularly if \overline{m} is chosen conservatively and if the size of the overall set is large.

3 Partitioned Set Reconciliation

We can reduce the computational complexity of set reconciliation by adopting a divide-and-conquer approach. This approach works by recursively partitioning the space of all possible bitstrings, continuing until a partition is reached on which BASIC-RECON succeeds with a prescribed bound

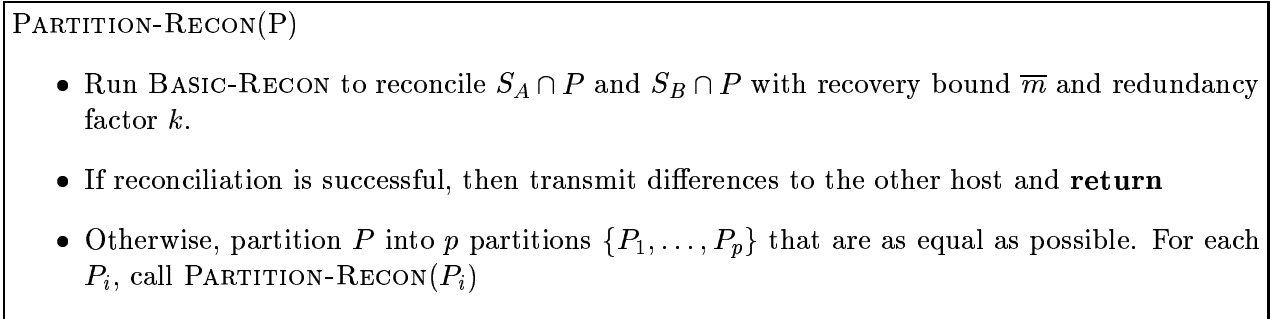


Figure 1: PARTITION-RECON

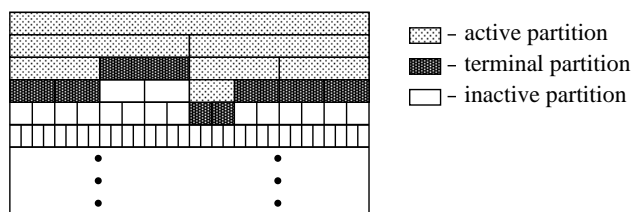


Figure 2: An example run of PARTITION-RECON with partitioning factor $p = 2$.

\overline{m} on the number of differences in the partition. Since the value of \overline{m} is fixed, this avoids the high computational cost of running BASIC-RECON with large values of \overline{m} .

More formally, a *partitioning* of a set P is defined as a sequence $\{P_i\}$ of non-intersecting subsets of P such that $\cup_i P_i = P$. Given two sets $S_A, S_B \subset P$, one can reconcile S_A and S_B by separately reconciling $S_A \cap P_i$ and $S_B \cap P_i$ for each i , and then taking the union of the recovered sets.

The protocol PARTITION-RECON, shown in Figure 1, is based on a recursive partitioning scheme. The protocol has three parameters: the *partitioning factor* p , which is the number of equally-sized partitions created at each level, the recovery bound \overline{m} , and the redundancy factor k . PARTITION-RECON is initially called on the full set of all b -bit bitstrings.

Figure 2 illustrates one execution of PARTITION-RECON. Each box corresponds to a different partition, and the sub-partitions of each partition are arranged below it. The *level* of a partition P is the number of partitions containing it. Thus, there is a single level-0 partition containing all possible bitstrings, p level-1 partitions in the row below that, p^2 level-2 partitions in the row below that, *etc.* A partition P is classified into one of the following three categories, depending on the size of the symmetric difference $m_P = |(S_A \oplus S_B) \cap P|$:

- A partition P is *active* if $m_P > \overline{m}$. In this case, PARTITION-RECON(P) will recursively invoke itself on the p sub-partitions of P .
- A partition P is *terminal* if it is the root partition or the sub-partition of an active partition, and $m_P \leq \overline{m}$. In this case, the recursion of PARTITION-RECON terminates with a successful invocation of BASIC-RECON.
- A partition P is *inactive* if it is descended from a terminal partition.

3.1 Worst-Case Analysis

A run of PARTITION-RECON results in an invocation of PARTITION-RECON(P) if and only if partition P is active or terminal. Unfortunately, it is possible for the symmetric difference of reconciling sets to distribute very awkwardly among partitions, leading to a large number of active and terminal partitions in the worst case. The following theorem bounds the number of active and terminal partitions that can occur given an arbitrary distribution of elements.

Theorem 1 *For a fixed value \overline{m} , PARTITION-RECON is invoked recursively at most*

$$1 + \frac{m}{\overline{m}} p \lceil b \log_p(2) \rceil \quad (5)$$

times.

Proof: Define $I(s, m)$ to be the number of invocations of PARTITION-RECON, where s is the size $|P|$ of the partition being reconciled, and m is the size of the symmetric difference of $S_A \cap P$ and $S_B \cap P$. The following recurrence bounds $I(s, m)$.

$$I(s, m) \leq 1 + \begin{cases} 0 & \text{if } m \leq \overline{m} \\ \max_{\sum_i^p m_i = m} \left\{ \left[\sum_i I\left(\left\lceil \frac{s}{p} \right\rceil, m_i\right) \right] \right\} & \text{otherwise.} \end{cases} \quad (6)$$

In the first case of the recurrence, we know that BASIC-RECON will succeed, and so there will be no recursive invocations of PARTITION-RECON. The second case corresponds to a worst-case bound for how the differences between two sets are split when each host set is partitioned. Here, m_i represents the number of differences contained in the i -th partition. Note that implicit in the formulation of the second case is the observation that $I(s, m)$ is necessarily non-decreasing in s .

We proceed to prove the theorem by induction on s . For the moment, we only consider s that are integral powers of p . For the base case of the induction, note that $I(i, m) = 1$ for all $m \leq \overline{m}$, and that $m \leq s$. Our inductive hypothesis is:

$$I(s, m) \leq 1 + p(\log_p s) \frac{m}{\overline{m}}.$$

Consider the case of $I(ps, m)$, where $s > \overline{m}$. If $m \leq \overline{m}$, the inductive case follows trivially. If $m \geq \overline{m}$, then from the recurrence in Equation (6), we get:

$$\begin{aligned} I(ps, m) &\leq 1 + \max_{\sum_i^p m_i = m} \left[\sum_{i=1}^p I(s, m_i) \right] \\ &\leq 1 + \max_{\sum_i^p m_i = m} \sum_{i=1}^p \left[1 + p(\log_p s) \frac{m_i}{\overline{m}} \right] \\ &= 1 + p \left(1 + (\log_p s) \frac{m}{\overline{m}} \right) \\ &\leq 1 + p(1 + (\log_p s)) \frac{m}{\overline{m}} \\ &= 1 + p(\log_p ps) \frac{m}{\overline{m}} \end{aligned}$$

This completes the induction and proves the theorem for values of s that are powers of p . By monotonicity of $I(s, m)$, we can conclude that for arbitrary values of s , the $I(s, m)$ is bounded by

$$1 + p(\lceil \log_p s \rceil) \frac{m}{\overline{m}}.$$

Plugging in $s = 2^b$ gives the desired bound. ■

The computation and communication complexities of PARTITION-RECON, expressed in the following two corollaries, follow directly from Theorem 1.

Corollary 1 *Excluding the cost of generating checksums, PARTITION-RECON has a worst-case computation time of*

$$\Theta(m\bar{m}^2 b^2 \frac{p}{\log p})$$

for a fixed redundancy factor k .

Proof: The computational cost of any give invocation of PARTITION-RECON is $\Theta(b\bar{m}^3 + bmk)$. Multiplying by the number of invocations from Theorem 1 yields the desired bound. ■

Note that a complete reconciliation with PARTITION-RECON also requires the computation of checksums. We address the complexity of computing these in Section 4.

Corollary 2 *The worst-case communication complexity of PARTITION-RECON is $\Theta(m\frac{p}{\log p}b^2)$.*

Proof: The communication complexity of a single invocation of PARTITION-RECON(P) is simply the size of a checksum, plus an extra bit for the second host to return its status, either failure or success. Multiplied by the bound on the number of invocations from Theorem 1, this yields the stated result. ■

For fixed p , the communication complexity is $O(mb^2)$, which is significantly worse than the $O(mb)$ complexity of BASIC-RECON. The above worst-case analysis also allows us to bound the probability of a failed invocation of BASIC-RECON by $\epsilon_k \cdot (1 + m/\bar{m}p \lceil b \log_p(2) \rceil)$. The redundancy factor k must be chosen to make this probability acceptably small.

3.2 Expected-Case Analysis

The expected case of PARTITION-RECON is much better than the worst-case analysis of Theorem 1 would suggest. In order to do an expected case analysis, we assume that the elements of the sets being reconciled are chosen uniformly and at random. This is a reasonable approximation if the data being reconciled is in fact hashed-values of otherwise arbitrary data, as discussed in Section 1. Even if hashed values are not needed, hashing can be used to determine which partitions a given bitstring belongs in, making the bitstrings effectively random.

The performance of PARTITION-RECON depends on the distribution of active and terminal partitions, and in this section we provide some analysis of that distribution. Using that analysis we then bound both the round complexity and the computational complexity of PARTITION-RECON.

The following lemma estimates the probability that a given partition is active.

Lemma 1 *Let P be some level- k partition, and let S be a collection of randomly-chosen b -bit bitstrings. The probability that P contains more than \bar{m} elements from S is no more than*

$$\text{full}(|S|, k) \stackrel{\text{def}}{=} \sum_{i=\bar{m}+1}^{|S|} \text{binom}(i; |S|, \frac{1}{p^k} + \frac{1}{2^b})$$

where $\text{binom}(k; n, p)$ is the usual shorthand for $\binom{n}{k} p^k (1-p)^{n-k}$ and $p \geq 2$ is the partitioning factor.

Proof: It can be shown through a simple recurrence that a level- k partition must have cardinality at most $\lceil 2^b/p^k \rceil + 1$. The lemma then follows from elementary probability. \blacksquare

Define a level k to be active if any partition P on that level is active. The following theorem estimates the number of active levels, which can in turn be used to estimate the number of rounds required by the protocol.

Theorem 2 *The expected number of active levels is at most:*

$$\left\lceil \left(1 + \frac{1}{\bar{m}}\right) \log_p \left(\frac{2em}{\bar{m} + 1}\right) \right\rceil + 3 \leq 2 \log_p \left(\frac{m}{\bar{m} + 1}\right) + O(1),$$

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

Proof: By Lemma 1, the probability that a level- k partition P is active is bounded above by $\text{full}(m, k)$. Thus, we may apply the union bound to get

$$\begin{aligned} \Pr(\text{level } k \text{ is active}) &= \Pr(\exists \text{ an active level-}k \text{ partition}) \\ &\leq p^k \text{full}(m, k). \end{aligned} \tag{7}$$

We can further bound $\text{full}(m, k)$ using some well known bounds on the tail of the binomial distribution [1, equation (6.9) and Theorem 6.2]:

$$\text{full}(m, k) = \sum_{i=\bar{m}+1}^m \text{binom}(i; m, \frac{1}{p^k} + \frac{1}{2^b}) \leq \left(\frac{2em}{(\bar{m} + 1)p^k}\right)^{\bar{m}+1}. \tag{8}$$

Therefore,

$$\Pr(\text{level } k \text{ is active}) \leq p^k \text{full}(m, k) \leq \left(\frac{2em}{\bar{m} + 1}\right)^{\bar{m}+1} \frac{1}{p^{k\bar{m}}}. \tag{9}$$

When the bound of Equation (9) is greater than 1, we can trivially replace it with 1. This replacement occurs when

$$\begin{aligned} \left(\frac{2em}{(\bar{m} + 1)p^k}\right)^{\bar{m}+1} &\geq 1 \\ \log \left(\left(\frac{2em}{\bar{m} + 1}\right)^{\bar{m}+1}\right) &\geq k\bar{m} \\ k &\leq \left\lfloor \frac{\bar{m} + 1}{\bar{m}} \log \left(\frac{2em}{\bar{m} + 1}\right) \right\rfloor \stackrel{\text{def}}{=} \kappa \end{aligned} \tag{10}$$

$$\tag{11}$$

We can therefore write:

$$\mathcal{E}(\text{active levels}) = \sum_{k=0}^{b \log_p(2)} \mathcal{E}(\text{level } k \text{ is active}) \quad (12)$$

$$\leq \sum_{k=0}^{b \log_p(2)} \min \left(1, \left(\frac{2em}{\bar{m}+1} \right)^{\bar{m}+1} \frac{1}{p^{k\bar{m}}} \right) \quad (13)$$

$$= \sum_{k=0}^{\kappa} 1 + \left(\frac{2em}{\bar{m}+1} \right)^{\bar{m}+1} \cdot \sum_{k=\kappa+1}^{b \log_p(2)} \left(\frac{1}{p^{k\bar{m}}} \right) \quad (14)$$

$$\leq \kappa + 1 + \left(\frac{2em}{\bar{m}+1} \right)^{\bar{m}+1} \cdot \frac{1}{p^{m\kappa}(p^m - 1)} \quad (15)$$

$$\leq \kappa + 1 + \frac{1}{p^m - 1} \quad (16)$$

$$\leq \kappa + 3 \quad (17)$$

Substituting the definition of κ into the last line above gives the desired bound. \blacksquare

The following theorem bounds the number of active and terminal partitions.

Theorem 3 *The expected number of active and terminal partitions is bounded above by*

$$\frac{8e(p+1)m}{\bar{m}+1}. \quad (18)$$

Proof: We first estimate the number of active partitions. By linearity of expectation we can conclude that

$$\mathcal{E}(\# \text{ active partitions}) = \sum_{k=0}^{b \log_p(2)} \mathcal{E}(\# \text{ active partitions at level } k). \quad (19)$$

Using Equation (8), we can bound the expected number of active partitions at a given level as follows:

$$\mathcal{E}(\# \text{ active partitions at level } k) \leq p^k \text{full}(m, k) \leq \left(\frac{2em}{\bar{m}+1} \right)^{\bar{m}+1} \frac{1}{p^{k\bar{m}}} \quad (20)$$

Note, however, that the expected number of active partitions at level k cannot be more than p^k , which is the total number of partitions at that level. Thus, We can replace the bound in (20) by p^k whenever

$$\begin{aligned} \left(\frac{2em}{\bar{m}+1} \right)^{\bar{m}+1} \frac{1}{p^{k\bar{m}}} &\geq p^k \\ (\bar{m}+1) \log_p \left(\frac{2em}{\bar{m}+1} \right) &\geq (\bar{m}+1)k \\ k &\leq \left\lfloor \log_p \left(\frac{2em}{\bar{m}+1} \right) \right\rfloor \stackrel{\text{def}}{=} \kappa \end{aligned}$$

We can therefore write:

$$\begin{aligned}
\mathcal{E}(\# \text{ active partitions}) &\leq \sum_{k=0}^{\kappa} p^k + \sum_{k=\kappa+1}^{b \log_p(2)} \left(\frac{2em}{\bar{m}+1} \right)^{\bar{m}+1} \frac{1}{p^{k\bar{m}}} \\
&\leq \frac{p^{\lfloor \kappa \rfloor} - 1}{p-1} + \left(\frac{2em}{\bar{m}+1} \right)^{\bar{m}+1} \cdot \sum_{k=\kappa+1}^{b \log_p(2)} \frac{1}{p^{k\bar{m}}} \\
&\leq \frac{p}{p-1} \left(\frac{2em}{\bar{m}+1} \right) + \left(\frac{2em}{\bar{m}+1} \right)^{\bar{m}+1} \cdot \frac{1}{p^{\bar{m}\kappa}(p^{\bar{m}}-1)} \\
&\leq \frac{p}{p-1} \left(\frac{2em}{\bar{m}+1} \right) + \left(\frac{2em}{\bar{m}+1} \right) \cdot \frac{p^{\bar{m}}}{(p^{\bar{m}}-1)} \\
&\leq \left(\frac{2em}{\bar{m}+1} \right) \left(\frac{p}{p-1} + \frac{p^{\bar{m}}}{p^{\bar{m}}-1} \right) \\
&\leq \left(\frac{8em}{\bar{m}+1} \right)
\end{aligned}$$

Since there are at most p times as many terminal partitions as there are active partitions, multiplying by $p+1$ yields the desired bound. \blacksquare

Complexity Results. We now have the tools to compute the expected round complexity and computational complexity of PARTITION-RECON. There is no interdependence between the execution of PARTITION-RECON(P) for different partitions P at the same level. Thus, all executions of PARTITION-RECON(P) at the same level can be executed in a single round. The round-complexity of PARTITION-RECON is therefore just the number of levels that have some active or terminal partitions, which is one more than the number of active levels. Thus,

Corollary 3 *The expected number of rounds needed by PARTITION-RECON for reconciliation is at most*

$$2 \log_p \left(\frac{m}{\bar{m}+1} \right) + O(1).$$

The overall amount of communication and computation needed by PARTITION-RECON is determined by the total number of active and terminal partitions, as bounded in Theorem 3. The following two corollaries follow from Theorem 3 and Equations (4) and Table 1.

Corollary 4 *The expected number of bits transmitted by PARTITION-RECON is at most*

$$8emp(b+1) + \frac{8emkp(b+1)}{\bar{m}+1} \in O(mb).$$

If the redundancy and partitioning factors k and p are fixed.

Corollary 5 *Ignoring the cost of computing checksums, the expected amount computation required by PARTITION-RECON is*

$$O(mp b(\bar{m}^2 + k)).$$

Thus, in the expected case, PARTITION-RECON has computational and communication complexity that is linear in the number of differences between reconciling sets, and requires only logarithmically many rounds of communication.

4 Incremental Partitioned Set Reconciliation

A straightforward implementation of PARTITION-RECON would simply compute the necessary checksums at the time of reconciliation. This simple approach, however, can be problematic if a given host engages in multiple reconciliations on the same evolving data set. In that case, much of the cost of computing the checksums will be repeated unnecessarily at each reconciliation. This section shows how to avoid this problem by computing checksums incrementally as elements are inserted and deleted from the set. In this approach, all checksums that might be necessary in a run of PARTITION-RECON are stored in a data structure called a *partition tree* that can be updated incrementally.

A partition tree is a simple p -ary tree with two kinds of nodes: *internal nodes*, and *leaf nodes*. Each node in a partition tree corresponds to a different partition P . Consider the partition tree for some set S . An internal node corresponding to partition P contains a checksum cs_P of the elements $S \cap P$, along with pointers to the p nodes representing the sub-partitions of P . A leaf node corresponding to partition P contains only the set $S \cap P$. Figure 3 shows a portion of a partition tree with branching factor $p = 3$. Internal nodes are drawn as rectangles and leaf nodes as ovals. To the right of each node is a label of the corresponding partition.

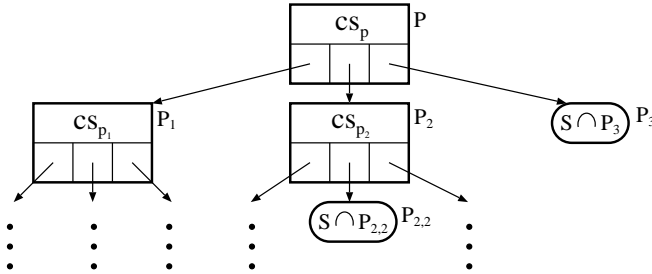


Figure 3: A trinary partition tree for set S of 3-bit bitstrings.

A key invariant maintained by the partition tree is that there is an internal node for every partition P such that $|P \cap S| > \bar{m}$. This ensures that it is always possible to obtain the checksums necessary for reconciliation. Thus, when a bitstring β is added to S , the partition tree must be updated accordingly. In particular, it is necessary to descend the tree, using `add_element` to update the checksums for all the internal nodes representing partitions to which β belongs, finally inserting β into the set of the leaf node whose partition contains β . Similarly, when removing a bitstring β from S , `delete_element` must be used to update the appropriate checksums. Also, if the number of bitstrings in a leaf node is raised above \bar{m} , it must be converted into an internal node, and children created for it. Similarly, if the number of elements in a leaf node is brought below $\bar{m} + 1$, it must be converted to an internal node, and its children destroyed.

4.1 Expected-Case Analysis

In this section, we compute the expected cost of inserting and deleting a random element from a partition tree populated by a random set S . We say that a node in the tree contains a bitstring β if the partition corresponding to that node contains β . The *depth* of a partition tree at β is the number of nodes in the partition tree containing β . In order to compute the expected cost of insertion and deletion, we first need to compute the expected depth of the partition tree at some random bitstring β .

Theorem 4 *Let S be a random collection S of b -bit bitstrings, and consider a partition tree representing S . The expected depth of the partition tree at S is bounded above by:*

$$\left\lceil \log_p \left\lfloor \frac{|S|}{\overline{m}} \right\rfloor \right\rceil + 4 \quad (21)$$

Proof: In order to bound the expected depth at β , we first compute the expected number of partitions P containing β such that $|P \cap S| > \overline{m}$. Every such partition is represented by an internal node in the partition tree, and there is one leaf node below those internal nodes. Thus, the expected depth is 1 plus the expected number of internal nodes.

Let X_k be a random variable that is 1 if the level- k partition containing β has more than \overline{m} elements, and 0 otherwise. Clearly, $\mathcal{E}(X_k)$ is bounded above by 1. Lemma 1 indicates that $\mathcal{E}(X_k)$ is bounded above by $\text{full}(|S|, k)$. Moreover, for any set S and level k , there are at most $\lfloor |S|/\overline{m} \rfloor$ internal node partitions P (i.e. with $|P \cap S| > \overline{m}$). Thus, $\mathcal{E}(X_k)$ can be bounded as follows.

$$\mathcal{E}(X_k) \leq \min \left[1, \frac{\lfloor |S|/\overline{m} \rfloor}{p^k} \right]$$

Note that for $k \leq \kappa \stackrel{\text{def}}{=} \lfloor \log_p(\lfloor |S|/\overline{m} \rfloor) \rfloor$, the above bound is equal to 1.

If X is a random variable representing the number of internal nodes containing β then, by linearity of expectation,

$$\begin{aligned} \mathcal{E}(X) &\leq \sum_{k=0}^{b \log_p(2)} \mathcal{E}(X_k) \\ &\leq \sum_{k=0}^{\kappa} 1 + \sum_{k=\kappa+1}^{b \log_p(2)} \frac{\lfloor |S|/\overline{m} \rfloor}{p^k} \\ &\leq \kappa + 1 + \lfloor |S|/\overline{m} \rfloor \sum_{k=\kappa+1}^{b \log_p(2)} \frac{1}{p^k} \\ &\leq \kappa + 1 + \lfloor |S|/\overline{m} \rfloor \frac{1}{p^\kappa} \frac{1}{p-1} \\ &\leq \kappa + 1 + \frac{p}{p-1} \\ &< \lfloor \log_p(\lfloor |S|/\overline{m} \rfloor) \rfloor + 3. \end{aligned}$$

Adding one to the above bound on the number of active nodes yields the desired bound on the expected depth of the partition tree. ■

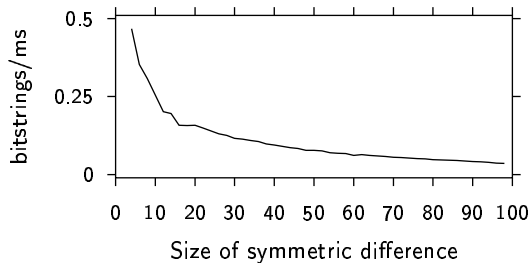


Figure 4: Speed of BASIC-RECON.

From Theorem 4, we can conclude that an insertion or deletion will require an expected number of no more than $\lfloor \log_p(|S|/\bar{m}) \rfloor + 3$ updates to internal nodes, plus 1 update to a leaf node. Each of the internal node updates has complexity $O(b\bar{m} + bk)$ as per Table 1, and the leaf node update has complexity of $\Theta(b\bar{m} + p)$. The following theorem thus follows.

Theorem 5 *The expected computational complexity of inserting or deleting an element to/from a partition tree representing set $|S|$ is*

$$\Theta(b(\bar{m} + k) \log_p(|S|/\bar{m}) + p).$$

For constant k and p , this reduces to

$$\Theta(\bar{m}(\log |S| - \log \bar{m})).$$

5 Experimental Results

We implemented the PARTITION-RECON algorithm using the partition-tree data structure of Section 4 to precompute the necessary checksums, and ran a series of experiments to test the performance of our implementation. This section presents some of the data from those experiments.

All our experiments have the following in common:

- The same redundancy factor $k = 1$ is used for every invocation of BASIC-RECON.
- Each displayed data point is the average over 10 runs.
- The cost of creating the partition tree is not included in the experimental results.
- The sets being reconciled have 10,000 elements in common, with each host having an extra $m/2$ elements not shared with the other host.
- Experiments were run between two 700MHz Pentium-III machines, connected by a 10Mb/s Ethernet.

We are mainly concerned with the following two performance metrics:

Redundancy: The total number of bits sent by the protocol divided by mb , the size of the symmetric difference.

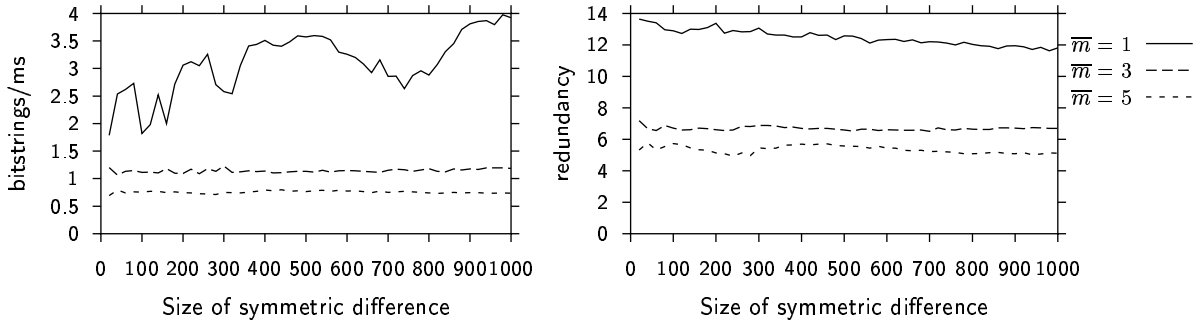


Figure 5: The first graph shows the speed of PARTITION-RECON plotted against the size m of the symmetric difference. The second graph shows redundancy plotted against m . \bar{m} is the size of the threshold used in invocations of BASIC-RECON, and $p = 4$ is the branching factor of the partitioning.

Speed: The number of bitstrings reconciled divided by the amount of time taken (in milliseconds) by the reconciliation. Because our experimental setup is largely computation-bound, the speed mostly reflects the computational complexity, but is also affected by the redundancy and the number of round-trips required by the protocol.

Before looking at the results for PARTITION-RECON, it is useful to consider the speed of the BASIC-RECON protocol by itself. Figure 4 shows the speed of BASIC-RECON for different values m of the symmetric difference. Since the computational complexity of BASIC-RECON is cubic in m , the speed drops off rapidly.

The first graph in Figure 5 shows the speed of PARTITION-RECON for different values of \bar{m} . Unlike BASIC-RECON, the speed is roughly constant in m , as suggested by the bounds of Section 3.2. Interestingly, for $\bar{m} = 1$ and 2, the performance seems to improve somewhat as m grows, which is better than our asymptotic bounds would suggest. Note also that reducing \bar{m} increases the speed, again in line with the bounds of Section 3.2. The large jump in speed between $\bar{m} = 3$ and $\bar{m} = 1$ comes from the fact that BASIC-RECON becomes much more efficient for $\bar{m} = 1$, due to the existence of simplified algorithms for interpolating and finding the roots of degree-1 polynomials.

The second graph in Figure 5 plots the redundancy of PARTITION-RECON for different values of \bar{m} . Note that the redundancy is roughly constant in m , showing that the communication complexity does indeed grow linearly in the size of the symmetric difference. The graphs in Figure 5 together show that by changing \bar{m} , it is possible to trade off redundancy for speed. The improvement in redundancy for larger values of \bar{m} is due to the fact that the redundancy of BASIC-RECON improves for larger values of \bar{m} . This in turn is largely due to the overhead associated with the redundancy factor k , which is more significant for smaller values of \bar{m} .

Figure 6 shows the performance of PARTITION-RECON for different values of p . Again, there is a tradeoff between redundancy and speed. Note that the performance oscillates in m . This is because the protocol performs optimally when the number of differences fits into an even number of levels. Larger branching factors lead to larger costs when the number of differences is stuck between levels.

Figures 7 and 8 show that increasing both p and \bar{m} reduces the number of round-trips. Note that increasing p from 2 to 4 has a large effect on the number of round-trips, with only a small effect on the redundancy, as shown in Figure 6.

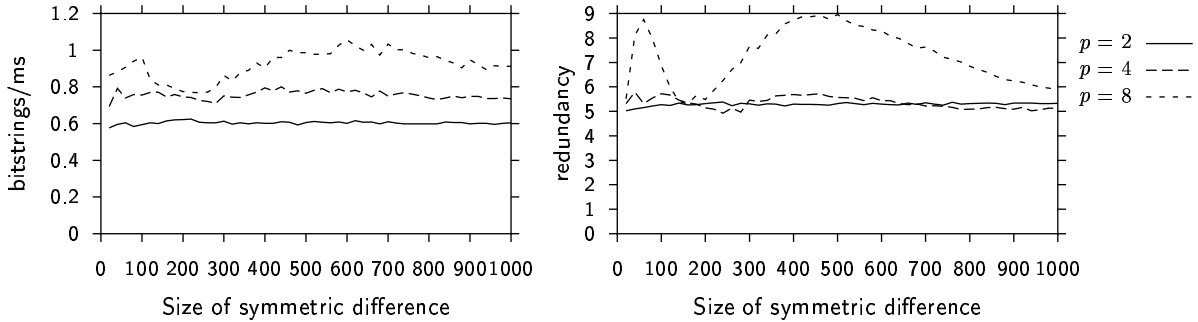


Figure 6: These graphs show the performance of PARTITION-RECON for different values of the partitioning factor p . The first graph shows the speed of PARTITION-RECON plotted against the size m of the symmetric difference. The second graph shows redundancy plotted against m . $\bar{m} = 5$ is the size of the threshold used in invocations of BASIC-RECON.

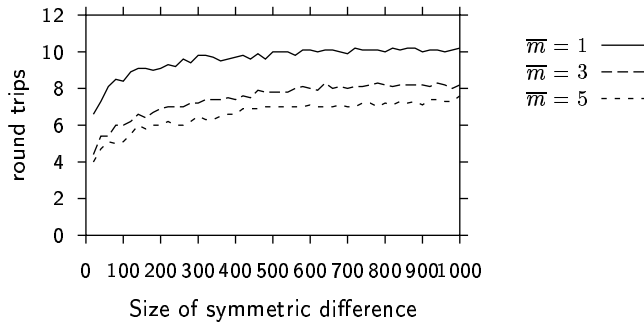


Figure 7: These graphs show how the number of round-trips changes with \bar{m} for partitioning factor $p = 4$.

6 Conclusion

We have presented a new algorithm PARTITION-RECON for set reconciliation that improves upon the existing BASIC-RECON algorithm. PARTITION-RECON operates by recursively partitioning the set of possible bitstrings and using BASIC-RECON as a subroutine on those partitions. Our analysis shows that the expected computational complexity is linear, as opposed to cubic for BASIC-RECON, and the communications complexity of PARTITION-RECON is within a small factor of that for BASIC-RECON. We also provide a data structure that allows for incremental maintenance of the data needed to do reconciliation, so that there is no need for a host to do a linear scan over its set before reconciling. We have implemented and tested PARTITION-RECON, and the algorithm performs well. We believe these algorithms can be effective in greatly increasing the performance, and therefore the feasibility, of anti-entropy based gossip protocols.

References

- [1] CORMEN, T., LEISERSON, C., AND RIVEST, R. *Introduction to Algorithms*. MIT Press, 1990.
- [2] DEMERS, A., GREENE, D. H., HAUSE, C., IRISH, W., AND LARSON, J. Epidemic algorithms for replicated database maintenance. In *Proceedings of the Sixth Annual ACM Symposium on*

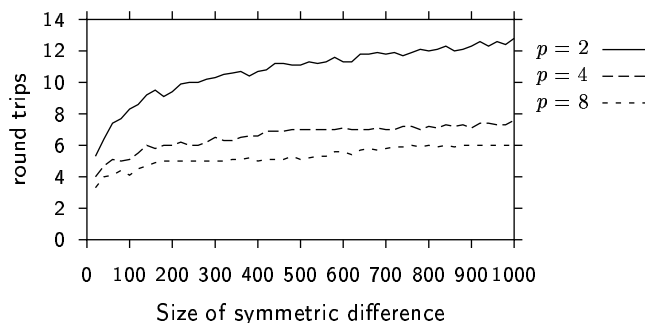


Figure 8: These graphs show how the number of round-trips changes with the partitioning factor p for $\bar{m} = 5$

- Principles of Distributed Computing* (Vancouver, British Columbia, Canada, August 1987), no. 6, ACM, pp. 1–12.
- [3] GOLDING, R. *Weak-Consistency Group Communication and Membership*. PhD thesis, UC Santa Cruz, December 1992. Published as technical report UCSC-CRL-92-52.
- [4] GUO, K., HAYDEN, M., RENESSE, R. V., VOGELS, W., AND BIRMAN, K. P. GSGC: An efficient gossip-style garbage collection scheme for scalable reliable multicast. Tech. rep., Cornell University, December 1997.
- [5] HAYDEN, M., AND BIRMAN, K. Probabilistic broadcast. Tech. rep., Cornell University, 1996.
- [6] KARPOVSKY, M., LEVITIN, L., AND TRACHTENBERG, A. Connections between data reconciliation and generalized error-correcting codes. *IEEE International Symposium on Info. Theory* (June 2001). recent result.
- [7] KARPOVSKY, M., LEVITIN, L., AND TRACHTENBERG, A. Data verification and reconciliation with generalized error-control codes. *39th Annual Allerton Conference on Communication, Control, and Computing* (July 2001).
- [8] KARPOVSKY, M., LEVITIN, L., AND TRACHTENBERG, A. Data verification and reconciliation with generalized error-control codes. *IEEE Trans. on Info. Theory* (2001). submitted.
- [9] LADIN, R., LISKOV, B., SHRIRA, L., AND GHEMAWAT, S. Providing high availability using lazy replication. *ACM Transactions on Computer Systems* 10, 4 (1992), 360–391.
- [10] MINSKY, Y., TRACHTENBERG, A., AND ZIPPEL, R. Set reconciliation with nearly optimal communication complexity. Tech. Rep. TR1999-1778, TR2000-1796, TR2000-1813, Cornell University, 2000.
- [11] MINSKY, Y., TRACHTENBERG, A., AND ZIPPEL, R. Set reconciliation with nearly optimal communication complexity. In *International Symposium on Information Theory* (June 2001), p. 232.
- [12] TERRY, D. B., THEIMER, M. M., PETERSEN, K., DEMERS, A. J., SPREITZER, M. J., AND HAUSER, C. H. 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), no. 22, ACM, pp. 172–183.

- [13] TRACHTENBERG, A., AND STAROBINSKI, D. Towards global synchronization. *Large Scale Networks workshop* (March 2001). <http://ana.lcs.mit.edu/sollins/LSN-Workshop/papers/>.
- [14] TRACHTENBERG, A., STAROBINSKI, D., AND AGARWAL, S. Fast PDA synchronization using characteristic polynomial interpolation. *Proc. INFOCOM* (June 2002). to appear.
- [15] VAN RENESSE, R., MINSKY, Y., AND HAYDEN, M. A gossip-style failure detection service. In *Middleware '98: IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing* (1998), N. Davies, K. Raymond, and J. Seitz, Eds., Springer Verlag, pp. 55–70.