Efficient Reconciliation of Unordered Databases

Ari Trachtenberg*
Digital Computer Laboratory
University of Illinois at Urbana-Champaign
arit@alum.mit.edu

Yaron Minsky*,†
Department of Computer Science
Cornell University
yminsky@cs.cornell.edu

November 9, 1999

Abstract

We consider the problem of reconciling two unordered databases whose contents are related. Specifically, we wish to determine the mutual difference of these databases with a minimum communication complexity. This type of problem arises naturally in the context of gossip protocols. We analyze two instances of the reconciliation problem, a client-server model and a more general peer-to-peer model, and provide interactive solutions for both. For the former instance, we also provide a simple one-message reconciliation algorithm, based on elementary symmetric polynomials, which has an almost optimal communication complexity. Finally, we examine several applications of these results, including the resource discovery problem.

*The authors are listed alphabetically by first name.

†Supported in part by ARPA/RADC grant F30602-96-1-0317, AFOSR grant F49620-94-1-0198, 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 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. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright annotation thereon.

1 Introduction

In this paper we consider a data transmission problem that arises naturally in the context of gossip protocols. In a gossip protocol, each host periodically chooses a companion with which to exchange information. With a sequence of these exchanges, information can be spread quickly throughout a network. Such protocols are used in a variety of systems to reliably and efficiently distribute information among a collection of networked hosts.

The information distributed by such a protocol is generally an unordered set of data. The data elements could contain a variety of different kinds of information, such as entries in a bibliographic database [6], USENET postings [2], broadcast messages and information about the stability of those messages [6, 7, 10], the names of hosts participating in the system and the names of failed hosts [6, 22, 8], or information about the location of resources [21].

The number of data entries that need to be exchanged in a given information exchange is generally small compared to the size of the entire database known to the participating hosts. It is therefore natural to ask whether it is possible to reduce the number of bits transmitted in each exchange. A plausible approach to this question would be to use timestamps to aid in discovering what data elements a given host is missing. In the context of gossip protocols, however, data elements can be updated in any order and at unpredictable times. Such considerations reduce the usefulness of timestamp schemes.

Another plausible approach, and one used in practice, is for one host to send to another the hash values of each of the elements in its database. This allows the latter host to determine which elements differ and the necessary elements can then be sent [6]. This solution is only helpful when the data elements are significantly larger then the size of the hash value, which is not always the case (e.g. when the data are IP addresses). Moreover, even in the case where the hash values are smaller than the data elements, there is still the question of how to transmit those hash values efficiently.

We formalize the *set reconciliation problem* as follows. Consider a pair of hosts p and q that each have a database of b-bit vectors, stored in sets S_p and S_q respectively. The goal is for p to learn the contents of the set difference $\delta_q = S_q \setminus S_p$, and for q to learn the contents of $\delta_p = S_p \setminus S_q$ while communicating as few bits as possible (i.e. minimizing the *communication complexity* of the algorithm). The *multi-set reconciliation problem* is a straightforward generalization in which the databases are stored as multi-sets with possibly duplicated vectors.

There are applications for the set reconciliation problem outside of gossip protocols. In particular, we believe it is useful in many situations where there is erasure or other corruption affecting two or more replicas of a database. As a concrete example, consider a system of client workstations whose disks are maintained as mirrors of an image maintained on a server. When a workstation's disk becomes corrupted in some way (i.e. important files are modified and/or erased), the common practice is to simply reload the image onto the machine in question. In situations where network bandwidth is a critical resource, however, it would be desirable to efficiently discover what files are corrupted and repair or replace just those, without having to send the full image. If for each file we take the hash of the file's pathname and the file contents, then the problem of figuring out which hash values are not shared between a client and the server is an instance of the set reconciliation problem.

Variants of the reconciliation problem have been considered at great length in the literature. Abelson [1] introduced the notion of interactive communication, considering a continuous model in which two hosts attempt to compute a value of a "smooth" function whose parameters are shared between them. Yao [23] considered a discrete restriction to the computation of Boolean functions

and analyzed communication complexity for various deterministic and probabilistic models. In [14] and [17], two person communication problems using a pointer-jumping model are considered, where each host has a list of pointers to the other's list and the task is to follow a pointer chain until the k-th pointer.

Finally, a large body of relevant work on interactive communication problems is detailed in [3, 4, 12, 15] and the many citations therein. These results mostly focus on theoretical bounds for exchange of information about two related random variables. The work most closely related to ours is that of Orlitsky [16], who considers—from an information-theoretic perspective—the reconciliation of binary strings of a prescribed edit distance apart. Orlitsky analyzes and bounds how much communication is needed for this string reconciliation, leaving a concrete construction of a corresponding protocol as his main open problem.

In Section 2 we consider the problem of reconciling two databases where it is assumed that one host's database is a subset of the other's. We call this the *client-server* reconciliation problem. We describe a single-message (i.e. non-interactive) algorithm for reconciling multi-set databases with nearly optimal communication complexity and practical computational complexity. We also describe a computationally faster interactive algorithm and give bounds for its communication complexity.

In Section 3 we consider the more general *peer-to-peer* reconciliation problem. Here neither database is assumed to be a subset of the other. We provide two interactive algorithms based on hash functions to solve this problem. In Section 4 we examine some extensions of our reconciliation algorithms, an in Section 5 we demonstrate an application of our results to the resource discovery problem [8]. Finally, we present conclusions and future research directions in Section 6.

2 Client-Server Reconciliation

From an information-theoretic perspective, we can construct a lower bound on the communication complexity of the set reconciliation problem by considering the case in which each host clairvoyantly (i.e. without communication) determines which vectors the other host is missing. In the general, peer-to-peer case, transmission of the missing vectors demands that host p discern $|\delta_q|$ vectors among $2^b - |S_p|$ possibilities and symmetrically for host q, giving an information-theoretic communication lower bound of

$$\lg\left[\binom{2^b - |S_p|}{|\delta_q|} + \binom{2^b - |S_q|}{|\delta_p|}\right] \tag{1}$$

bits. In the case of multi-sets, missing vector possibilities range over the entire space resulting in a lower bound of $b(\delta_p + \delta_q)$ bits.

In this section, we consider the special case of client-server reconciliation. Let s denote the server, and c denote the client. Assume that the server's database S_s is a superset of the client's database S_c, and define $e = |S_s \setminus S_c|$ to be the number of b-bit vectors that the client must acquire from the server. We can drop the second term in the log in Equation (1) (since we assume it is known to both sides that S_c is a subset of S_s). This gives us the following bound on the number of bits required to solve the client-server set reconciliation problem.

$$\lg\left[\binom{2^b - |S_c|}{e}\right] \;=\; \lg\left[\binom{2^b - |S_s| + e}{e}\right] \tag{2}$$

The bound for the number of bits required for the client-server multi-set reconciliation problem is be. When S_s is significantly smaller than the set of possible length-b bitstrings, the bound for set

reconciliation is close to the bound of be that holds for multi-set reconciliation. In particular, if $S_s < 2^{b-1}$, then

$$(b - 1 - \lg e)e \leq \lg \left[\binom{2^b - |S_s| + e}{e} \right] \leq be. \tag{3}$$

Below, we present algorithms whose performance approaches these information-theoretic lower bounds.

2.1 Single message communication

The information-theoretic lower bound on communication complexity of multi-set reconciliation can be achieved trivially in one special case — specifically, when the server has exactly one more vector than the client.

Algorithm 2.1 *When $e = 1$, the client can reconcile with the server using b bits of communication as follows:*

- The server computes the parity sum of its vectors, p_s and sends this to the client.
- The client computes his own parity sum p_c.
- The client determines the missing vector as $p_s \oplus p_c$, where $\oplus$ denotes bit-wise exclusive or.

In Algorithm 2.1, the client obtains from the server the parity sum $p_s \oplus p_c$ of the data in $S_s \setminus S_c$. When $e = 1$, the parity sum is equal to the single member of $S_s \setminus S_c$. When $e > 1$, the sum of the missing vectors provides some information about those vectors, but not enough to allow the missing vectors to be recovered.

We can generalize the approach of Algorithm 2.1 to the case that $e > 1$ by using the elementary symmetric polynomials in the place of a parity sum. The i-th elementary symmetric polynomial $\sigma_i(x_1, x_2, x_3, \ldots, x_n)$ is the sum of all possible products of i terms chosen among the parameters. Thus, the first three elementary symmetric polynomials are:

$$\begin{aligned}
\sigma_1(x_1, x_2, x_3, \ldots, x_n) &= x_1 + x_2 + x_3 + \ldots + x_n \\
\sigma_2(x_1, x_2, x_3, \ldots, x_n) &= x_1x_2 + \ldots + x_1x_n + \ldots x_2x_3 \ldots + x_2x_n + \ldots + x_{n-1}x_n \\
\sigma_3(x_1, x_2, x_3, \ldots, x_n) &= x_1x_2x_3 + x_1x_2x_4 + x_1x_2x_5 \\
&\quad + \ldots + x_{n-2}x_{n-1}x_n
\end{aligned}$$

We use the notation $\sigma_i(T)$ to denote the i-th elementary symmetric polynomial of all the elements of set T and define $\sigma_0(T) = 1$ for any set T. Our algorithm for the $e > 1$ case rests on the following easily-proven convolution property of elementary symmetric polynomials.

Lemma 2.1 *Given sets S, C, and D, where $C \subseteq S$ and $D = S \setminus C$,*

$$\sigma_k(S) = \sum_{0 \leq i \leq k} \sigma_i(C)\sigma_{k-i}(D)$$

Algorithm 2.2 *When $e \leq \frac{|S_c|}{2}$, reconcile client data set S_c with server data set S_s as follows:*

- *The server sends size $|S_s|$ to the client, which computes $e = |S_s| - |S_c| = |S_s \setminus S_c|$.*
- *For all $1 \leq i \leq e$, the client reads from the server*

$$\mathsf{server_sum}_i = \sigma_i(S_s).$$

- *For all $1 \leq i \leq e$, the client computes*

$$\mathsf{client_sum}_i = \sigma_i(S_c).$$

- *The client uses the values $\mathsf{server_sum}_i$ and $\mathsf{client_sum}_i$ together with Lemma 2.1 to iteratively compute, for all $1 \leq i \leq e$ the symmetric polynomials $\sigma_i(D)$.*
- *The client solves for the missing vectors, which are solutions of the following equation in z:*

$$\prod_{x \in D}(z - x) = \sum_{i=0}^{|D|}(-1)^i \sigma_i(D) z^{|D|-i} = 0$$

Algorithm 2.2 generalizes Algorithm 2.1 to the case $e > 1$ and works whenever the client size is at least half as large as the server size. We shall denote the difference set between client and server by $D = S_s \setminus S_c$. Note that Algorithm 2.2 requires two extra messages in order to determine the value of e. However, if e, or an upper bound on e is known *a priori*, then Algorithm 2.2 can be reduced to a single-message protocol.

From a communication perspective, computing the various sums over the ring of integers is inefficient, since the symmetric polynomials would evaluate to integers of longer bit-length than the vectors in the original data sets. Therefore, in order to minimize communication complexity, all operations are performed over a finite field $\mathbb{F}_p$ for some prime $p > 2^b$, where b is the vector bit-length.[1] Including the cost of determining e, the server only needs to transmit $\lceil e \lg(p) \rceil + b$ bits to the client. Since, by Bertrand's Postulate [9, p. 343], there is always at least one prime number between 2^b and 2^{b+1}, we know that $\lg(p) \leq b + 1$. This gives the following communication bound, which is within $b + e$ bits of the information-theoretic optimum.

Theorem 2.1 *Algorithm 2.2 reconciles server set S_s and client set S_c using at most*

$$be + (b + \phi) \tag{4}$$

bits of communication, where ϕ is some real between 1 and e, depending on the size of the prime modulus p.

The computational complexity of Algorithm 2.2 is quite tractable. There are two bottlenecks in the algorithm: computation of the symmetric polynomials and root extraction at the end. The computation of the symmetric polynomials can be amortized over insertions in the server database,

[1] Performing computations over $\mathbb{F}_{2^b}$ would give the optimal communication complexity. However, our algorithm will require the use of factorization algorithms, the best of which [11] are known only over $\mathbb{F}_p$.

since

$$\sigma_i(S \cup \{x\}) = \sigma_i(S) + x * \sigma_{i-1}(S). \tag{5}$$

Thus, in order to maintain the values of e symmetric polynomials, the server needs only $O(e)$ time per insertion for an overall running time of $O(|S|e)$.

Moreover, the client's problem of finding roots of a polynomial over $\mathbb{F}_p$ has been well studied [13, 18, 19]. Kaltofen and Shoup's baby step/giant step technique [11] can factor this polynomial in expected time $O(e^{1.82} \log p)$. Thus, Algorithm 2.2 provides an efficient, non-interactive method for client-server reconciliation in practice. Example 1 demonstrates this algorithm for a simple reconciliation.

Example 1 *Consider the server set $S_s = \{1, 2, 3, 4, 5, 6\}$ and the client set $S_c = \{2, 4, 6\}$ for 3-bit integers. We use the prime number $p = 11$ as the communication modulus.*

After determining that $e = 3$, based on the server's set size, the client will read from the server:

$$\begin{aligned} \mathsf{server_sum}_1 &= \sigma_1(S_s) \equiv 10 \pmod{11} \\ \mathsf{server_sum}_2 &= \sigma_2(S_s) \equiv 10 \pmod{11} \\ \mathsf{server_sum}_3 &= \sigma_3(S_s) \equiv 9 \pmod{11} \end{aligned}$$

The client computes its own sums:

$$\begin{aligned} \mathsf{client_sum}_1 &= \sigma_1(S_c) \equiv 1 \pmod{11} \\ \mathsf{client_sum}_2 &= \sigma_2(S_c) \equiv 0 \pmod{11} \\ \mathsf{client_sum}_3 &= \sigma_3(S_c) \equiv 4 \pmod{11} \end{aligned}$$

Now the client uses Lemma 2.1 to compute:

$$\begin{aligned} \sigma_1(D) &= \sigma_1(S_s) - \sigma_1(S_c) \equiv 9 \pmod{11} \\ \sigma_2(D) &= \sigma_2(S_s) - \sigma_2(S_c) - \sigma_1(S_c)\sigma_1(D) \equiv 1 \pmod{11} \\ \sigma_3(D) &= \sigma_3(S_s) - \sigma_3(S_c) - \sigma_1(S_c)\sigma_2(D) - \sigma_2(S_c)\sigma_1(D) \equiv 4 \pmod{11} \end{aligned}$$

The client then constructs the equation

$$z^3 - 9z^2 + 1z - 4 \equiv 0 \pmod{11}$$

whose solutions are precisely $z \equiv 1, 3, 5 \pmod{11}$.

Algorithm 2.2 has the advantage of being non-interactive, since, after determining or bounding e, the server does not need to make use of any information from the client. Moreover, the communication complexity of Algorithm 2.2 is in no way sensitive to the size of the server's data set, only to its difference from the client set.

2.2 Interactive Algorithm

In this section we present an interactive reconciliation algorithm (Algorithm 2.3) that improves on the computational complexity of Algorithm 2.2 at the expense of higher computational complexity. For simplicity, we describe only a set (as opposed to multi-set) reconciliation algorithm. The generalization to multi-set reconciliation is straightforward.

Algorithm 2.3 *After comparing sizes to determine $e = |S_s \setminus S_c|$, server s and client c can reconcile their data by each running* reconcile(S, b, e) *with S chosen to be their data set (i.e. either S_s or S_c).*

reconcile(S, b, e)
if $e = 0$
return S
if $e = 1$
return the result of running Algorithm 2.1 on the remaining bits
else

- Partition S into sets A and B such that A contains all vectors whose b-th bit is 0, and B contains all vectors whose b-th bit is 1.
- Compute $|A| \mod (e+1)$ and send this information to the other party. We denote the server's value a_{server}, the client's value a_{client}, and

$$\text{numzero} = (a_{\text{server}} - a_{\text{client}}) \mod (e+1).$$

- **return** the union of reconcile$(A, b-1, \text{numzero})$ and reconcile$(B, b-1, e - \text{numzero})$

Algorithm 2.3 is a divide-and-conquer algorithm. Specifically, when $|S_s - S_c| = e > 1$ each host partitions its set into two subsets, based on the set's projection onto a certain bit position. The hosts exchange the sizes of their respective subsets and continue the recursion until either their subsets match in size, meaning that they must be equal, or else they differ by one, and Algorithm 2.1 is used to determine the missing vector. Figure 2.2 shows a small example of Algorithm 2.3 at work. The correctness of Algorithm 2.3 follows from the way in which the reconcile procedure breaks down the reconciliation process into two independent subproblems. Note that both the client and the server have the same value for e, so the recursive descent occurs identically at both hosts. The algorithm terminates because each recursive call decreases the parameter b. If b reaches zero then e must be 0 or 1, which is a terminating condition of the recursion.

The bound of Theorem 2.2 on the communication complexity of Algorithm 2.3 depends only on b, the vector bit-size, and e, the difference in size between the client and server sets.

Theorem 2.2 *Algorithm 2.3 needs at most $2be + 2b$ bits of communication to reconcile a client and server whose sets differ by $e = |S - C|$ b-bit vectors.*

Proof The initial determination of e requires $2b$ communication bits. To determine the rest of the communication complexity, define $C(b, e)$ to be the maximum total number of bits transmitted by both sides in a call to reconcile(S, b, e) for any set S. From the definition of reconcile, we can see that $C(b, e)$ will have the following recursive structure, corresponding to the recursive structure of the reconcile procedure:

$$C(b,e) \leq \begin{cases} 0 & \text{if } e = 0, \\ -\infty & \text{if } e > 0 \text{ and } b = 0, \\ b & \text{if } e = 1 \text{ and } b > 0, \\ \max\limits_{k=0\ldots e} \{C(b-1,k) + C(b-1,e-k) + 2\lceil \lg(e+1) \rceil\} & \text{otherwise.} \end{cases} \tag{6}$$

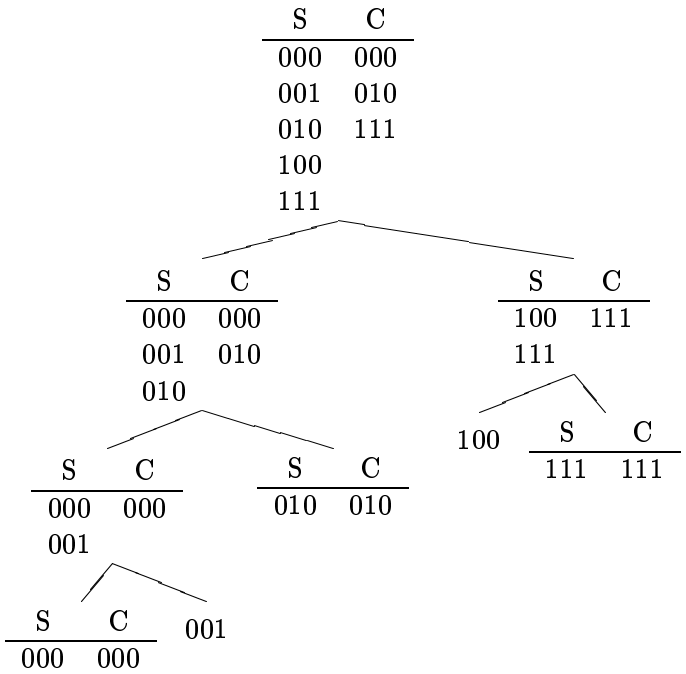

Figure 1: The recursion tree for Algorithm 2.3 on a small example.

The case where $e > 0$ and $b = 0$ is artificially set to $-\infty$ to simplify the proof. Since this case does not occur, the corresponding value of $C(b, e)$ can be chosen arbitrarily.

We prove the desired bound by induction on b for an arbitrary, fixed e. The first three (base) cases of (6) clearly fit Theorem 2.2. To show the inductive step, we assume Theorem 2.2 holds for all $b' < b$. Then,

$$\begin{aligned} C(b,e) &\leq C(b-1,k') + C(b-1,e-k') + 2\left\lceil \lg(e+1)\right\rceil, \text{ for some k'} \\ &\leq k'(2b-2) + (e-k')(2b-2) + 2\left\lceil \lg(e+1)\right\rceil \qquad (7) \\ &= 2be + (\left\lceil 2\lg(e+1)\right\rceil - 2e) \\ &\leq 2be \qquad (8) \end{aligned}$$

Step (7) is derived using the inductive hypothesis and base cases, and step (8) is based on the concavity of the logarithm. ∎

The following lemma presents a lower bound on the worst-case communication complexity of the reconcile procedure in Algorithm 2.3.

Lemma 2.2 *When e is a power of two,*

$$C(b,e) \geq \left[2\sum_{i=0}^{\lg e-1}\left(2^i\left\lceil \lg\left(\frac{e}{2^i}+1\right)\right\rceil\right)\right] + [2(b-\lg e)e] + e + 2b \qquad (9)$$

The bound (9) is reached when server and client differ by the set of vectors

$$D_e = \left\{\left\langle v_{[2]} \mid 0^{b-e} \mid l\right\rangle \text{ s.t. } v < 2^{\lg(e)-1} \text{ and } l \in \{0,1\}\right\} \qquad (10)$$

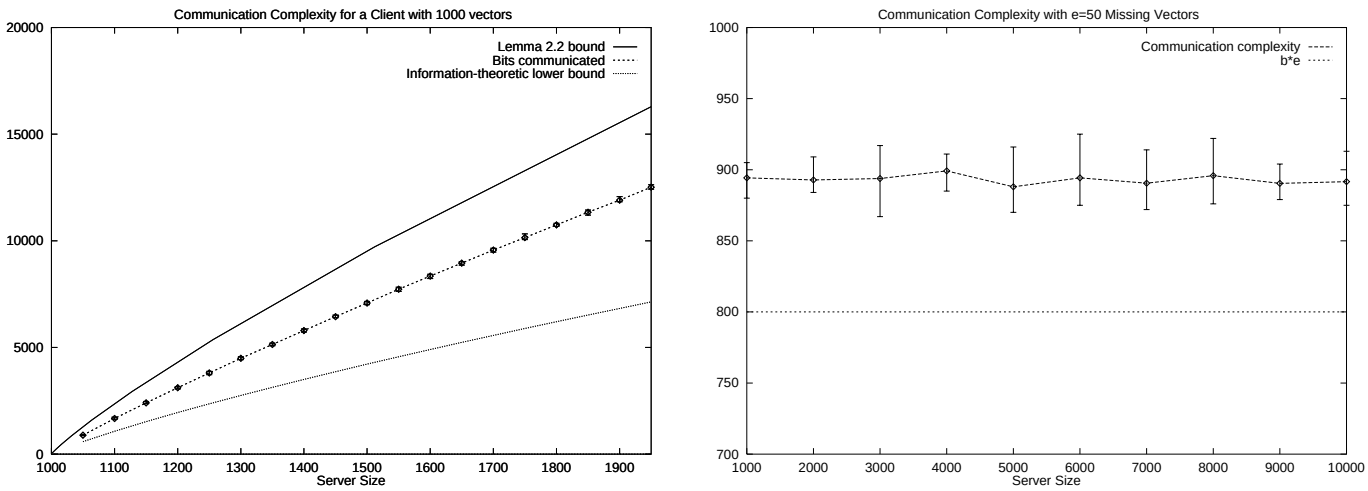

Figure 2: Simulation data for Algorithm 2.3 compared to the theoretical upper bound given by explicitly computing (6). The left hand figure shows the communication complexity for a fixed client of 1000, 16-bit vectors. In the right hand figure, we set $e = 50$ and show that the communication complexity is not affected by scaling the server size.

As per convention, $\langle \cdot | \cdot \rangle$ denotes concatenation, and $v_{[2]}$ denotes the binary form of v. For example, when $e = 8$ the desired difference set is given by:

$$\begin{aligned} D_8 = \{ & 00\ 000 \ldots\ 0\ 0, \quad 00\ 000 \ldots\ 0\ 1 \\ & 01\ 000 \ldots\ 0\ 0, \quad 01\ 000 \ldots\ 0\ 1 \\ & 10\ 000 \ldots\ 0\ 0, \quad 10\ 000 \ldots\ 0\ 1 \\ & 11\ 000 \ldots\ 0\ 0, \quad 11\ 000 \ldots\ 0\ 1\} \end{aligned}$$

When e is not a power of two, we can clearly use $C(b,t)$ as a lower bound on $C(b,e)$ by choosing t to be the largest power of 2 smaller than e.

The terms of (9) correspond to the components of the vectors of D_e. Specifically, the first term corresponds to the first third of each vector, $\langle v_{[2]} \rangle$ which causes Algorithm 2.3 to recursively split into two sub-problems, each with half the missing vectors of the parent problem. The second third of each vector, $\langle 0^{b-e} \rangle$, causes algorithm to recursively call itself with exactly the same number (i.e. 2) of missing vectors. There are $\frac{e}{2}$ branches of recursion, each of length $(b - \lceil \lg e \rceil)$, which involve the communication of $2\lceil \lg 3 \rceil$ bits (since we have to communicate modulo $2+1=3$ if there are 2 missing vectors). Thus, the second third of the vector contributes a communication complexity of

$$2\lceil \lg 3 \rceil (b - \lceil \lg e \rceil)\frac{e}{2} = 2(b - \lceil \lg e \rceil)e.$$

Finally, when we reach the end of a vector of D_e, we send precisely one bit to distinguish each missing vector, giving an additional communication complexity of e bits. The $2b$ term at the end of (9) corresponds to the cost of initially determining the value e. ∎

Though we have thus far been concerned mostly with communication complexity, the computational complexity of Algorithm 2.3 is actually also quite tolerable, as is seen in Theorem 2.3.

Theorem 2.3 *The running time for Algorithm 2.3 on data set S is at most*

$$(b+e)|S| + O(be) \quad \subseteq \quad O\left((b+e)|S|\right)$$

Proof First of all, we run Algorithm 2.1 exactly once for each missing vector. Since there are e missing vectors, this part of the computation will require $e(|S|+1)$ computations: for each missing vector we compute $|S|$ parity-sums and one additional sum to compare the server and client. Secondly, the partitioning step will, during the course of the algorithm, examine each vector a maximum of b times, corresponding to the maximum depth of recursion of the algorithm. This part of the computation will, therefore, require at most $b|S|$ comparisons. Finally, a constant number of computations per procedure call remain for the various other operations (e.g. computing numzero). Since our recursion tree has maximum depth b and maximum breadth $2e$, we have $O(be)$ computations for this stage. The result follows from the sum of these three components. ∎

In most practical systems, data is not sent bit by bit but rather in fixed-size packets. For this reason, it is often useful to minimize the number of packets, rather than bits, communicated between client and server. Consider the case where all transmissions are sent as t-bit packets. We can trivially adapt Algorithm 2.3 to this case by compacting the information in several layers of the recursion tree into one t-bit transmission. Such a modified algorithm would require transmission of at most

$$2\left\lceil \frac{t}{\lg e} \right\rceil be + 2b \tag{11}$$

bits, corresponding to roughly $2b\frac{e}{\lg e}$ packets. This is better than a straightforward transmission of the entire server data set S when e is small relative to $\frac{|S|}{t}$.

3 Peer-to-Peer Reconciliation

In this section, we consider the case where there are two hosts, p and q, each with its own database set (S_p and S_q respectively), but with no assumed relationship between the host sets. As in the client-server case, the generalization to multi-sets is straightforward, but we omit it for simplicity.

To use a divide-and-conquer approach in reconciling data sets, it is essential to be able to determine efficiently that two sizable data sets are in fact equal. To do this, Algorithm 2.3 relied on the fact that, if $S \subseteq S'$, then $|S| = |S'|$ iff $S = S'$. Thus, by transmitting only the size of a set it is possible to test for equality. Without the subset property, however, we need a different way to test for equality. In the following we present an algorithm that uses hash functions for this purpose.

A *hash function* $H : \Sigma^* \longrightarrow \Sigma^k$ is a function that accepts an arbitrary string over an alphabet Σ and returns a fixed-length string over the same alphabet. The difficulty of finding such a pair is bounded by h, the length of the returned string—if h is too small, then a sufficiently large number of strings can be examined such that a pair with the same hash value will be found with high probability. The following algorithms presume the existence of a hash function H with a length h such that

$$H(A) = H(B) \;\Rightarrow\; A = B. \tag{12}$$

holds with high probability. A case where $A \neq B$ and yet $H(A) = H(B)$ is called a *collision*.

Choosing an appropriate hash function is a delicate question. The probability that there will be a collision depends on the way in which A and B are chosen. In our case, A and B are determined by the sets S_p and S_q. If S_p and S_q were chosen truly randomly, then almost any hash function would work equally well. In practice S_p and S_q are not chosen randomly. A reasonable constraint would be to assume that the choice of reconciliation data is made by a computationally bounded

Algorithm 3.1 *Hosts p and q initiate a reconciliation by p calling* reconcile(S, b) *for their respective databases S of b-bit vectors.*

reconcile(S, b)
if $|S| = 0$ **then**
get missing vectors from the other party
else if the other party's set is empty
send S to other party.
else if our hash $H(S)$ equals the other party's hash H_{other}
return S
else
- Partition S into sets A and B such that A contains all vectors whose b-th bit is 0, and B contains all vectors whose b-th bit is 1.
- **return** the union of reconcile$(A, b-1)$ and reconcile$(B, b-1)$

adversary. In this case, we can ensure that Equation (12) holds with high probability by using a *collision-resistant* hash function.

A collision-resistant hash function H is a hash function for which it is computationally infeasible to find (with high probability) a pair of different strings A and B such that $H(A) = H(B)$. Though it is not known in theory whether or not collision-resistant hash functions exist there are hash functions that may be used in practice such as the Secure Hash Standard [5].

For the remainder of this section, we will simply assume that Equation (12) holds unconditionally.

Algorithm 3.1 is a direct analog of Algorithm 2.3. Like Algorithm 2.3, its communication complexity is independent on the sizes of the respective client and server sets.[2] Unfortunately, it is still impractical for many applications due to the large hash size h needed for reasonable collision-resistance. In the following, we denote the size of the symmetric difference of S_p and S_q by $e = |S_q \setminus S_p| + |S_p \setminus S_q|$.

Theorem 3.1 *The communication complexity of Algorithm 3.1, in bits, is given by*

$$be(4h + 1)$$

Proof The recursion tree for reconcile has maximum depth b, since we assume vectors of S are unique. It also has a maximum breadth $2e$, corresponding to one differing vector in each recursion subtree. This yields at most $2be$ calls to reconcile per party, each call transmitting h bits, for a total of $4beh$ transmitted bits. Moreover, each missing vector is transmitted exactly once, amounting to be transmitted bits. The net transmission is therefore $4beh + be$ bits. ∎

Algorithm 3.2 improves on the communications complexity of Algorithm 3.1, by partitioning the set of vectors according to the median of one host's set, rather than partitioning the underlying space evenly. This algorithm requires less communication when the database size S is small compared to the underlying space $\mathbb{F}_2^b$.

[2] The communication complexity is, however, dependent on the size of the underlying vector space, which ultimately bounds the set sizes. Thus, this complexity is *not* independent of database size in the multi-set case.

Algorithm 3.2 *Hosts p and q initiate a reconciliation by p calling* reconcile-active(S_p) *by q calling* reconcile-passive(S_q). *For termination p can send a flag to q when the initial call to* reconcile-active *returns.*

```
reconcile-active(S)
    if |S| ≤ 1
        Send ⟨S⟩ to the other party and receive ⟨S'⟩ from it
        return S ∪ S'
    else
        Get hash ⟨H_other⟩ of the other party's vectors
        if H_other is the hash of the empty set then
            Send ⟨S⟩ to the other party
            return S
        else if H_other = H(S) then
            Notify the other party that the vectors coincide
            return S
        else
            Send ⟨m⟩, the (lower) median of S, to the other party
            Denote the intervals lower-range= [0^b, m] and upper-range= (m, 1^b]
            return reconcile-active(S ∩ lower-range) ∪
                   reconcile-active(S ∩ upper-range)

reconcile-passive(S')
    upon receiving vectors ⟨S⟩ from the other party
        Send ⟨S' \ S⟩ to the other party
        return S' ∪ S
    upon receiving a request for a hash
        Send ⟨H(S')⟩
        if H(S') is the hash of the empty set then
            Receive ⟨S⟩ from the other party
            return S' ∪ S
        else if The other party reports that the vectors coincide
            return S'
        else Receive the (lower) median ⟨m⟩ from the other party
            Denote the intervals lower-range= [0^b, m] and upper-range= (m, 1^b]
                return reconcile-passive(S' ∩ lower-range) ∪
                       reconcile-passive(S' ∩ upper-range)
```

The correctness of Algorithm 3.2 follows from the following Theorem.

Theorem 3.2 *If hosts p and q make calls to* reconcile-active(S_p) *and* reconcile-active(S_q) *respectively, then both function calls will return $S_p \cup S_q$.*

Proof We proceed by induction over the size of S_p.

Base case: If $|S_p| \leq 1$ then p will send $\langle S_p \rangle$ to q, and q will respond with $\langle S_q \setminus S_p \rangle$ and then return $S_p \cup S_q$. Upon receiving the reply, p will return $S_p \cup (S_q \setminus S_p) = S_p \cup S_q$.

Induction Step: We assume that $|S_p| > 1$ and that the theorem holds for all sets of size less then $|S_p|$. Since $|S_p| > 1$, p requests the hash $H_{\text{other}} = H(S_q)$ from q. There are three possibilities for the set S_q:

$S_q = \emptyset$: This is the trivial case where $S_p \cup S_q = S_p$ and follows by inspection.

$S_q = S_p$: This is the trivial case where both procedures return upon discovering that their hashes are equal.

$S_q \neq S_p \ \cap \ S_q \neq \emptyset$: In this case p sends m, the (lower) median of S_p, to q. Define the intervals lower-range $= [0^b, m]$ and upper-range $= (m, 1^b]$. Then p and q call reconcile-active and reconcile-passive respectively on the sets $S_p \cap$ lower-range and $S_q \cap$ lower-range, and then on the sets $S_p \cap$ upper-range and $S_q \cap$ lower-range.

Since $|S_p| > 1$, it follows that $|S_p \cap \text{upper-range}|$ and $|S_q \cap \text{lower-range}|$ are less then $|S_p|$. Thus, we can invoke the inductive hypothesis to conclude that the first calls to reconcile-active and reconcile-passive return

$$(S_p \cap \text{lower-range}) \cup (S_q \cap \text{lower-range}) \ = \ (S_p \cup S_q) \cap \text{lower-range}.$$

Similarly, the second calls return $(S_p \cup S_q) \cap$ upper-range. The union of these two values is $S_p \cup S_q$, and both p and q return this value.

∎

Theorem 3.3 describes the communication complexity of this reconciliation scheme. Note that this scheme is not symmetric, so that it is best, if possible, to designate the smaller host as p (i.e. the active host).

Theorem 3.3 *Algorithm 3.2 needs to transfer at most*

$$2\left[e(\lg|S_p|)+1\right](b+h+2)+be \tag{13}$$

bits to reconcile hosts p and q. For typical usage when $h = b > 6$ and $|S_p| > 6$, this is at most $6\lg(|S_p|)be$ bits.

Proof The binary tree resulting from an execution of Algorithm 3.2 is balanced, and so has depth $\lceil\lg(|S_p|)\rceil$. We represent the execution tree by a binary tree where, for each node n, *range*(n) is the range to which the S_p and S_q have been constrained for the associated procedure calls.

We say a node n is *clean* if the two host sets agree on the associated range constraint, that is if $S_q \cap range(n) = S_p \cap range(n)$. Otherwise, we say n is *dirty*. The number of dirty *leaf* nodes must be no greater than e, the number of vectors at which the two sets differ. Since each dirty *internal* node must have at least one dirty child, the overall number of dirty nodes must be no greater than $e\lceil\lg(|S_p|)\rceil$. Moreover, reconcile-active is only called on a clean node when the node's parent is dirty or trivially when $e = 0$. Thus, the total number of reconcile-active calls on clean nodes is similarly bounded by $\max\{e\lceil\lg(|S_p|)\rceil, 1\}$.

We now make a count of the bits transmitted. We first consider the bits associated with sending the elements of $(S_q \setminus S_p) \cup (S_p \setminus S_q)$. Each such element is transmitted exactly once, costing be bits overall. Regarding the per-call communication costs, we note that at most $\max(b, h+1)+1$ additional bits are sent for a clean node, corresponding to communicating a hash value and notification flags. Similarly, the number of additional bits sent by a dirty node is no more than $h+b+2$, corresponding to communicating a hash value, the median, and notification flags.

This leads to the following bound

$$\max\{e\left\lceil \lg(|S_p|)\right\rceil, 1\}(h + b + \max(b, h + 1) + 3) + be, \tag{14}$$

which reduces to the stated result. ∎

The results of experimental simulations of Algorithm 3.2 are compared in Figure 3 to the upper bound in Theorem 3.3.

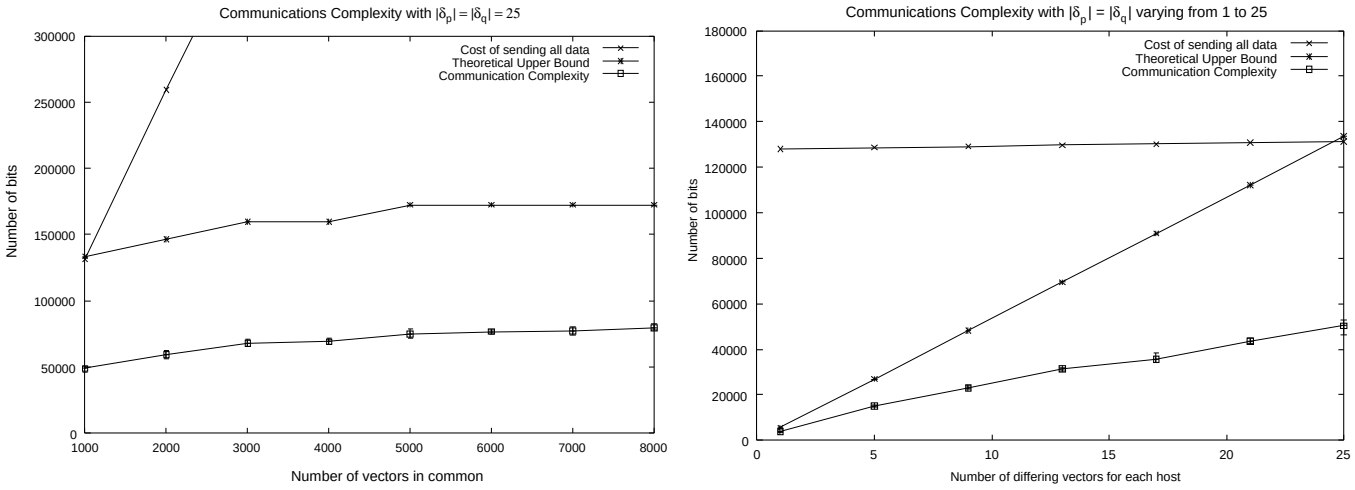

Figure 3: Simulation data for Algorithm 3.2 compared to the theoretical upper bound given by (13) and to the communication costs of simply sending all data. The left hand figure shows the communication complexity for a fixed client of 1000, 16-bit vectors, while in the right hand figure each host has 25 vectors that the other does not, and the number of common vectors varies.

4 Extensions

The algorithms in this paper generalize to some simple variations on the reconciliation problem. For example, these algorithms can be adapted to the problem of resolution, in which two hosts attempt to find the intersection of their respective data sets. Similarly, one might imagine a client reconciling [resolving] with many different data sets, provided the client maintains its original data set as it reconciles [resolves] with each host.

A related problem is the α-edits problem [16], which may be thought of as an ordered instance of the multi-set reconciliation problem. In this problem, a host p has a string S_p and another host has a string S_q; the strings are related in that one may be attained from the other using at most α insertions or deletions, collectively called *edits*. The goal is for p to learn q's string. The restricted version of this problem, in which only insertions are permitted, is called the α-insertions problem.

Algorithms 2.2 and 2.3 may be adapted for the α-insertions problem, and Algorithms 3.1 and 3.2 may be adapted for the α-edits problem. The one difficulty is that we must deal with the possibility of repeated characters in different locations of a string by extending the base alphabet. Moreover, the extra information used to disambiguate characters must be agreed upon by the hosts. This means that the characters in the strings in question must be augmented ahead of time, before the edits occur.

Finally, all of the algorithms in this paper stand to benefit from data compression. Specifically, they all have non-interactive data (e.g. the parity sums used to compute the missing vectors) that can be compiled into one long string that is compressed and sent at the end of the algorithm.

5 An Application

As mentioned in the introduction, gossip protocols can be used for disseminating the identities of the hosts participating in a system [6, 8, 22]. In this section, we will show how our peer-to-peer reconciliation protocols can be used to improve the communication complexity of one such protocol.

5.1 The resource discovery problem

For simplicity, we base our example on the model presented by Harchol-Balter, Leighton, and Lewin [8] for the resource discovery problem. Specifically, we define G to be a set of n hosts in a fully connected, synchronous network. We quantize the execution of algorithms by round, where a round is assumed to be long enough for each host on the network to contact another host and exchange information. For each host p in G, $\Gamma(p)$ is defined to be the subset of G consisting of the hosts known by p. Thus we can consider G to be a directed graph, with an edge from p to q iff $q \in \Gamma(p)$. We shall call this graph the *knowledge graph* of G.

The key assumptions made in [8] are as follows:

1. The knowledge graph is initially weakly-connected (i.e. the graph is connected when edge directions are ignored).

2. The network is static, meaning that no hosts are added or removed after startup.

They propose and analyze a protocol called Name-Dropper that is a simplified version of the protocol found in [22]. The Name-Dropper protocol works as follows: in every round each host p randomly picks another host q and sends to it the current value of $\Gamma(p)$. The analysis in [8] shows that with probability $1 - 1/n^{O(1)}$, the knowledge graph becomes the complete graph on G (i.e. each host learns about each other host) within $O(\lg^2 n)$ rounds. If the protocol terminates after $O(\lg^2 n)$ rounds, then the overall number of host names transmitted is bounded by $O(n^2 \lg^2 n)$.

5.2 Terminating case

We can reduce the number of host names sent in a run of Name-Dropper by using peer-to-peer reconciliation (i.e. Algorithm 3.2) instead of having the initiating host simply send the names of all of its known hosts. Algorithm 3.2 allows more information to be transferred with a smaller communication overhead than in the unmodified Name-Dropper algorithm. In particular, this algorithm permits q to learn the contents of $\Gamma(p)$ and p to learn the contents of $\Gamma(q)$. Thus, any edge in the knowledge graph that is present in a given run of Name-Dropper is also present in the analogous run of the modified protocol.

We now analyze the number of host names transmitted by r rounds of the modified protocol. For simplicity, we assume that the n host names have constant length b bits. Let h be the length of the hash-values used in the reconciliation algorithm. Consider a run of Algorithm 3.2 between hosts p and q, where $e_p = \Gamma(p) \setminus \Gamma(q)$, $e_q = \Gamma(q) \setminus \Gamma(p)$, and $e = e_p + e_q$. From Theorem 3.3, we can conclude that the number of bits sent is less than

$$e\ [2(\lg |S_p|)\ (b + h + 2) + b]\ +\ 2(b + h + 2),$$

where $|S_p|$ is bounded above by $\lg n$.

To simplify the analysis, we will account separately for the term linear in e (i.e. $e\,(2(\lg n)\,(b+h+2)+b)$), and the constant term (i.e. $2(b+h+2)$. The constant term is accrued at most once per reconciliation, and there are at most rn reconciliations for the r rounds. Thus, that component is bounded by $2rn(b+h+2)$. We charge part of the cost of the linear term to each host, according to the number of host names each host must learn. Thus, p will be charged $e_p\,[2(\lg n)\,(b+h+2)+b]$ and likewise q will be charged $e_q\,[2(\lg n)(b+h+2)+b]$.

Let e_p^α be the number of host names learned by host p in the α-th call of Algorithm 3.2, where index α spans the set of executions of Algorithm 3.2 over a given execution of the modified Name-Dropper protocol. Then the total cost accounted to host p is the sum

$$\sum_\alpha \left[e_p^\alpha\,[2(\lg n)(b+h+2)+b]\right] \;=\; \left[\sum_\alpha e_p^\alpha\right][2(\lg n)(b+h+2)+b]\,. \tag{15}$$

Since p never learns the same host's name twice, it follows that $\sum_\alpha e_p^\alpha \le n$. Thus, we have a bound of $n[2(\lg n)(b+h+2)+b]$ charged to each host. For all hosts together, this comes to $n^2[2(\lg n)(b+h+2)+b]$. If we add in the constant term, we get a bound of $n^2\,[2(\lg n)(b+h+2)+b]+2rn(b+h+2)$. If the protocol terminates after $r=O(\lg^2 n)$ rounds, we get $(2(\lg n)+b)n^2+n\,O(\lg^2 n)\,(b+h+2)$, which is $O(n^2\lg n)$. This improves the bound in [8] by a factor of $\lg n$ and is consistent with that paper's assertion that its results are within polylogarithmic factors of the optimal solution.

5.3 Steady state

The analysis of the number of host-names transmitted by Name-Dropper depends on the expectation that the algorithm terminates after $O(\lg^2 n)$ rounds. However, the common paradigm is for hosts to be introduced slowly into a network rather than all at once. In fact, other similar protocols [6, 22] are used to create a service to manage the ongoing addition and removal of hosts, and thus have no need to terminate. This leads to a longer execution which emphasizes the per-round cost of the protocol. As such, it seems appropriate to consider the ongoing cost of a protocol rather than simply its startup costs.

As a concrete example, consider the case where one host is introduced every round. Under such circumstance, Name-Dropper cannot possibly conclude in fewer than n rounds. The number of host names transmitted is therefore $O(n^3)$, since on the i'th round each of i hosts would transmit i transmissions. The modified Name-Dropper algorithm, however, still only has complexity $O(n^2\lg n)$. This translates to an average of $O(\lg n)$ host names transferred per host per round, as opposed to $O(n)$ host names in the unmodified Name-Dropper algorithm.

We leave as an open problem the analysis of the impact of set-reconciliation on protocols that allow for both the removal and addition of hosts [6, 22].

6 Conclusions and future research

We have considered the reconciliation of two unordered databases from the perspective of minimizing communication complexity. This perspective lends itself to the single-message and interactive algorithms that we have described in this paper. These algorithms efficiently reconcile two databases that have at least half of their elements in common; otherwise, the two databases do not share enough in common to make these reconciliations efficient.

Specifically, we have examined two instances of the reconciliation problem: client-server reconciliation, when one database is known to be a subset of the other, and peer-to-peer reconciliation,

when the two databases are arbitrary. For the first instance, we have presented a single-message algorithm that performs client-server reconciliation independently of the size of the server database, even for databases expressed as multi-sets with possibly repeated vectors. Moreover, for multi-set databases, the communication complexity of this algorithm is only a few bits worse than that of a clairvoyant server that determines, without any client input, and then optimally sends exactly the missing vectors to the client. We have also presented a comparable interactive algorithm with lower computational complexity. For the case of peer-to-peer reconciliation, we have presented two divide-and-conquer interactive algorithms based on different methods of dividing the data space. Finally, we have given, by means of example, a concrete application of these reconciliation algorithms to the problem of resource discovery.

Perhaps the most interesting addition to this work would be to find a way to extend the elegant use of symmetric functions in the non-interactive client-server solution (Algorithm 2.2) to the peer-to-peer case. There are also several natural extensions that we have not addressed. These include the problem of reconciling three or more sources and the question of how to apply the results of this paper to spurious error correction, where check bits may be lost in addition to data bits.

7 Acknowledgments

We are grateful to Ramin Takloo-Bighash, Edward Reingold, Fred Schneider, Lazar Trachtenberg, and Alexander Vardy for stimulating discussions. We would like to thank Tanya Berger-Wolf for suggesting the application to [8].

References

[1] Abelson, H. Lower bounds on information transfer in distributed computations. In *Proceedings of the 19th Annual Symposium on Foundations of Computer Science* (1978).

[2] Adams, R. RFC1036: Standard for interchange of USENET messages, December 1987.

[3] Ahlsweded, R., Cai, N., and Zhang, Z. On interactive communications. In *IEEE Transactions on Information Theory* (January 1997), vol. 43, pp. 22–37.

[4] ElGamal, A., and Orlitsky, A. Interactive data compression. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing* (1984), pp. 100–108.

[5] Secure hash standard. National Bureau of Standards FIPS Publication 180, 1993.

[6] Golding, R. A. *Weak-Consistency Group Communication and Membership.* PhD thesis, UC Santa Cruz, December 1992. Published as technical report UCSC-CRL-92-52.

[7] 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.

[8] Harchol-Balter, M., Leighton, T., and Lewin, D. Resource discovery in distributed networks. In *18th Annual ACM-SIGACT/SIGOPS Symposium on Principles of Distributed Computing* (Atlanta, GA, May 1999).

[9] Hardy, G., and Wright, E. *An Introduction to the Theory of Numbers.* Oxford University Press, 1954.

[10] Hayden, M., and Birman, K. Probabilistic broadcast. Tech. rep., Cornell University, 1996.

[11] Kaltofen, E., and Shoup, V. Subquadratic-time factoring of polynomials over finite fields. In *27th Annual ACM Symposium on Theory of Computing* (1995), vol. 9, pp. 398–406.

[12] Körner, J., and Orlitsky, A. Zero-error information theory. In *IEEE Transactions on Information Theory* (1998), vol. 44, pp. 2207–2229.

[13] Naudin, P., and Quittê, C. *Theoretical Computer Science*, vol. 191. Elsevier Science B.V., 1996, ch. Univariate Polynomial Factorization over Finite Fields, pp. 1–36.

[14] Nisan, N. Rounds in communication complexity revisited. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing* (1991), pp. 81–91.

[15] Orlitsky, A. Average-case interactive communication. In *IEEE Transactions on Information Theory* (July 1992), vol. 38, pp. 1534–1547.

[16] Orlitsky, A. Interactive communication of balanced distributions and correlated files. *SIAM Journal on Discrete Mathematics 6*, 4 (November 1993), 548–564.

[17] Papadimitriou, P. H., and Sipser, M. Communication complexity. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing* (1982), pp. 330–337.

[18] Shoup, V. Factoring polynomials over finite fields: Asymptotic complexity vs. reality. In *Proc. IMACS Symposium* (Lille, France, 1993).

[19] Shoup, V. A new polynomial factorization algorithm and its implementation. *Journal of Symbolic Computation 20* (1995), 363–397.

[20] Stinson, D. R. *Cryptography: Theory and Practice.* CRC press, 1995.

[21] van Renesse, R. Captain cook: A scalable navigation service. In preparation.

[22] 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.

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