Fast PDA Synchronization Using Characteristic Polynomial Interpolation

Ari Trachtenberg, David Starobinski, and Sachin Agarwal

Abstract—**Modern Personal Digital Assistant (PDA) architectures often utilize a wholesale data transfer protocol known as "slow sync" for synchronizing PDAs with Personal Computers (PCs). This approach is markedly inefficient with respect to bandwidth usage and latency, since the PDA and PC typically share many common records. We propose, analyze, and implement a novel PDA synchronization scheme (CPIsync) predicated upon recent information-theoretic research. The salient property of this scheme is that its communication complexity depends on the number of differences between the PDA and PC, and is essentially independent of the overall number of records. Moreover, our implementation shows that the computational complexity of CPIsync is practical, and that the overall latency is typically much smaller than that of slow sync. Thus, CPIsync has potential for significantly improving synchronization protocols for PDAs and, more generally, for heterogeneous networks of many machines.**

I. Introduction

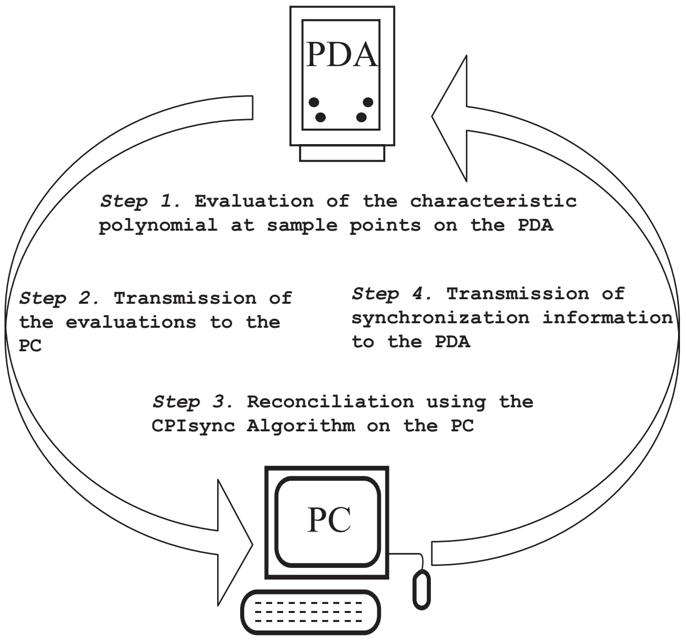
Much of the popularity of mobile computing devices and PDAs can be attributed to their ability to deliver information to users on a seamless basis. In particular, a key feature of this new computing paradigm is the ability to access and modify data on a mobile device and then to *synchronize* any updates back at the office or through a network. This feature plays an essential role in the vision of pervasive computing, in which any mobile device will ultimately be able to access and synchronize with any networked data.

Current PDA synchronization architectures, though simple, are often inefficient. With few exceptions, they generally utilize a protocol known as *slow sync* [1], which employs a wholesale transfer of all PDA data to a PC in order to determine differing records. This approach turns out to be particularly inefficient with respect to bandwidth usage and latency, since the actual number of differences is often much smaller than the total number of records stored on the PDA. Indeed, the typical case is where handheld devices and desktops regularly synchronize with each other so that few changes are made between synchronizations.

We propose to apply a near-optimal synchronization methodology based on recent research advances in fast set reconciliation [2, 3], in order to minimize the waste of network resources. Broadly speaking, given a PDA and a PC with data sets A and B, this new scheme can synchronize the hosts using one message in each direction of length $|A-B|+|B-A|$ (i.e. essentially independent of the size of the data sets A and B). Thus, two data sets could each have millions of entries, but if they differ in only ten of them, then each set can be synchronized with the other using one message whose size is about that of ten entries.

The key of the proposed synchronization algorithm is a translation of data into a certain type of polynomial known as the *characteristic polynomial*. Simply put, each reconciling host (*i.e.* the PDA and the PC) maintains its own characteristic polynomial. When synchronizing, the PDA sends sampled values of its characteristic polynomial to the PC; the number of samples must not be less than the number of differences between the two hosts. The PC then discovers the values of the differing entries by *interpolating* a corresponding rational function from the received samples. The procedure completes with the PC sending updates to the Palm, if needed. The worst-case computation complexity of the scheme is roughly cubic in the number of differences. A schematic of our implementation, which we call CPIsync for Characteristic Polynomial Interpolation-based Synchronization, is presented in Figure 1.

The authors are with the Department of Electrical and Computer Engineering at Boston University, Boston, MA. The work of Ari Trachtenberg is based, in part, upon work supported by the National Science Foundation under NSF Grant No. CCR-0133521.


PDA
Step 1. Evaluation of the characteristic polynomial at sample points on the PDA
Step 2. Transmission of the evaluations to the PC
Step 4. Transmission of synchronization information to the PDA
Step 3. Reconciliation using the CPIsync Algorithm on the PC
PC
Fig. 1. The overall scheme of the experiments done with the CPIsync algorithm.

We have implemented CPIsync on a Palm Pilot IIIxe, a popular and representative PDA. Our experimental results show that CPIsync performs significantly better (sometimes, by order of magnitudes) than slow sync in terms of latency and bandwidth usage. On the other hand, as the number of differences between hosts increase, the computational complexity of CPIsync becomes significant; thus, if two hosts differ significantly, wholesale data transfer becomes the faster method of synchronization. We present a simple numerical method for determining the threshold at which it becomes better to use wholesale data transfer than CPIsync. Thus, if the goal is to minimize the time needed to perform synchronization, then CPIsync should be used when the number of differences is below the threshold.

Otherwise, slow sync should be used. Note that the value of the threshold is typically quite large, making CPIsync the protocol of choice for many synchronization applications.

Another complication of CPIsync is that it requires a good *a priori* bound on the number of differences between two synchronizing sets. We describe two practical approaches for determining such a bound. In the first case, we propose a simple method that performs well for the synchronization of a small number of hosts (e.g. a PDA with two different PCs, one at work and one at home). In the second case, we make use of a probabilistic technique from [2] for testing the correctness of a guessed upper bound. If one guess turns out to be incorrect, then it can be modified in a second attempted synchronization, and so forth. The error of this probabilistic technique can be made arbitrarily small. We also show that the communication and time used by this scheme can be maintained within a small multiplicative constant of the communication and time needed for the optimal case where the number of differences between two hosts is known.

This paper is organized as follows. In the next section we begin with a review of the synchronization techniques currently used in the Palm OS computing platform and indicate their limitations. Thereafter, in Section III we establish the foundations of CPIsync, which are based on the theoretical results of [2]. Section IV provides technical details of our specific implementation of CPIsync on a Palm Pilot IIIxe. We also present experimental results for the case where a tight bound on the number of differences is known *a priori*. In Section V, we describe and evaluate the performance of a probabilistic technique that is used when a tight bound on the number of differences is not known *a priori*. We then discuss related work in Section VI and conclusions in Section VII.

II. BACKGROUND: THE PALM SYNCHRONIZATION PROTOCOL

In order to clearly and concretely explain the types of performance issues addressed in this paper, we describe next how data synchronization is implemented in the Palm OS architecture, one of the leading and state-of-the-art mobile computing platforms.

The Palm synchronization protocol, known as HotSync, relies on metadata that is maintained on both the handheld device and a desktop. The metadata consist of databases (Palm DBs) which contain information on the data records. A Palm DB is separately implemented for each application: there is one Palm DB for “Date Book” data records, another for “To Do” data records, and so forth. For each data record, the Palm DB maintains: a unique record identifier, a pointer to the record’s memory location, and status flags. The status flags remain clear only if the data record has not been modified since the last synchronization event. Otherwise the status flags indicate the new status of the record (*i.e.* modified, deleted, etc.).

The Palm HotSync protocol operates in either one of the following two modes: *fast sync* or *slow sync*. If the PDA device synchronizes with the same desktop as it did last, then the fast sync mode is selected. In this case, the device uploads to the desktop only those records whose Palm DB status flags have been set. The desktop then uses its synchronization logic to reconcile the device’s changes with its own. The synchronization logic may differ from one application to another and is implemented by so-called *conduits*. The synchronization process concludes by resetting all the status flags on both the device and the desktop. A copy of the local database is also saved as a backup, in case the next synchronization will be performed in slow sync mode.

If the fast sync conditions are not met, then a slow sync is performed. Thus, a slow sync is performed whenever the handheld device synchronized last with a different desktop, as might happen if one alternates synchronization with one computer at home and another at work. In such cases, the status flags do not reliably convey the differences between the synchronizing systems and, instead, the handheld device sends *all* of its data records to the desktop. Using its backup copy, the desktop determines which data records have been added, changed or deleted and completes the synchronization as in the fast sync case. An illustration of the fast sync and slow sync operation modes is given in Figure 2.

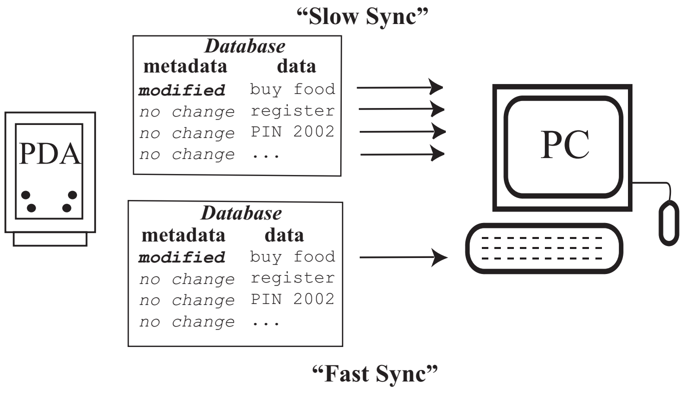


Fig. 2. The two modes of the Palm HotSync protocol. In the “slow sync” all the data is transferred. In the “fast sync” only modified entries are transferred between the two databases.

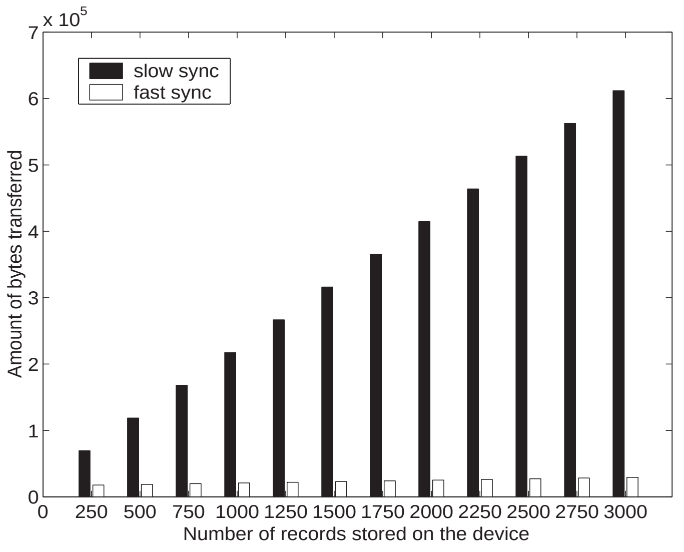


Fig. 3. A comparison of the communication complexities of slow sync and fast sync.

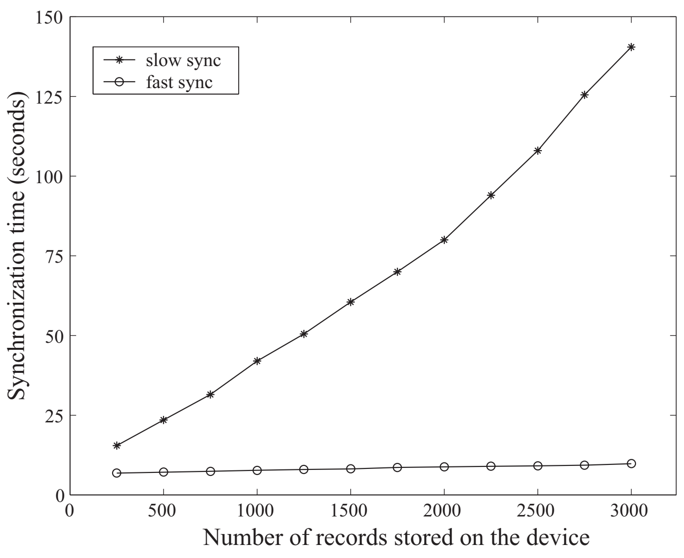


Fig. 4. A comparison of the time complexities of slow sync and fast sync.

It turns out that slow syncs are significantly less efficient than fast syncs, especially with respect to latency and use of bandwidth. In particular, the amount of communication and latency of slow syncs increase with the number of records stored in the device, independently of the number of differing records. Figures 3 and 4 illustrates this phenomenon on a Palm IIIxe PDA, as measured by a demo version of the Frontline Test Equipment software [4]. Specifically, these figures show the number of bytes transferred and the amount of time expended during similar slow sync and fast sync events. Our measurements are repeated with an increasing number of records on the device, but a fixed number of differences (*i.e.* ten); the records are all of the same size. In Figure 4 we see that the time needed to complete a slow sync grows roughly linearly with the number of records stored in the device, whereas for fast sync it remains almost constant. For the case of 3000 records, the duration of slow syncs exceeds 2 minutes, about 15 times longer than fast syncs. In fact, slow sync can require as long as 20 minutes for large, but practical, database sizes.

Figures 3 and 4 clearly show that slow syncs do not scale well with the amount of information stored on a device. Thus, the Palm synchronization model generally works well only in simple settings where users possess a single handheld device that synchronizes most of the time with the same desktop. However, this model fails in the increasingly common scenario where large amounts of data are synchronized among multiple PDAs, laptops, and desktops.

A seemingly plausible solution to this problem is to use timestamps or version control to aid in discovering what data elements a given host is missing. Unfortunately, the use of timestamps for synchronization can also result in inefficient communication, as depicted in Figure 5. In addition, timestamp protocols require each host to maintain information about each other host in the network, which does not scale well to multiple hosts and adapts poorly to a dynamic network in which hosts enter and leave regularly.

In summary, the key challenge to efficient PDA synchronization is a synchronization protocol whose communication complexity depends only on the number of differences between synchronizing systems, even when the conditions for a fast sync do not hold. In the next section, we present a family of such protocols based on characteristic polynomial interpolation.

III. Characteristic Polynomial Interpolation-based Synchronization

We formalize the problem of synchronizing two hosts' data as follows: given a pair of hosts A and B, each with a set of b-bit integers, how can each host determine the symmetric difference of the two sets (*i.e.* those integers held by A but not B, or held by B but not A) using a minimal amount of communication. Within this context, only the contents of the sets is important, but not their actual organization. Note also that the synchronized integers can generically encode all types of data. In [2] this formalization is called the *set reconciliation* problem. Natural examples of set reconciliation include synchronization of bibliographic data [5], resource availability [6, 7], data within gossip protocols [8, 9], or memos and address books. On the other hand, synchronization of edited text is not an example of set reconciliation because the structure of data in a file encodes information; for example, a file containing the string "a b c" is not the same as a file containing the string "c b a".

The set reconciliation problem is intimately linked to design questions in coding theory and graph theory [10, 11] from which several solutions exist. The following solution, which we have implemented on a PDA as described in Section IV, requires a nearly minimal communication complexity and operates with a reasonable computational complexity.

A. Deterministic scheme with a known upper bound

The key to the set reconciliation algorithm of Minsky, Trachtenberg, and Zippel [2, 3] is a translation of data sets into polynomials designed specifically for efficient reconciliation. To this end, [2] makes use of a characteristic polynomial $\chi_S(Z)$ of a set $S = \{x_1, x_2, \ldots, x_n\}$, defined to be:

$$\chi_S(Z) = (Z - x_1)(Z - x_2)(Z - x_3) \cdots (Z - x_n). \qquad (1)$$

If we define the sets of missing integers $\Delta_A = S_A - S_B$ and symmetrically $\Delta_B = S_B - S_A$, then the following equality holds

$$f(z) = \frac{\chi_{S_A}(z)}{\chi_{S_B}(z)} = \frac{\chi_{\Delta_A}(z)}{\chi_{\Delta_B}(z)}$$

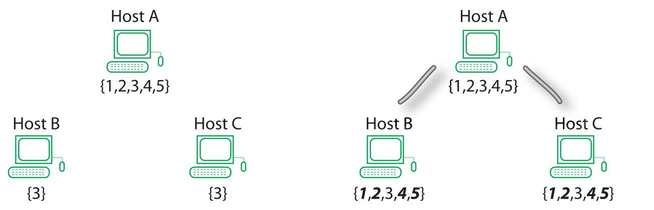


Fig. 5. Inefficiency of timestamps. Hosts B and C first synchronize so that each has item 3 (left figure). Thereafter hosts B and C independently synchronize with host A, noting the addition of items $1, 2, 4,$ and 5. When hosts B and C then re-synchronize, modification records require transmission of eight differences, marked in bold, whereas, in fact, there are none.

Protocol 1 Set reconciliation with a known upper bound $\overline{m}$ on the number of differences m. [3]

1. Hosts A and B evaluate $\chi_{S_A}(z)$ and $\chi_{S_B}(z)$ respectively at the same $\overline{m}$ sample points z_i, $1 \leq i \leq \overline{m}$.
2. Host B sends to host A its evaluations $\chi_{S_B}(z_i)$, $1 \leq i \leq m$.
3. The evaluation are combined at host A to compute the value of $\chi_{S_A}(z_i)/\chi_{S_B}(z_i) = f(z_i)$ at each of the sample points z_i. The points $(z_i, f(z_i))$ are interpolated by solving a generalized Vandermonde system of equations [2] to reconstruct the coefficients of the rational function

$$f(z) = \chi_{\Delta_A}(z)/\chi_{\Delta_B}(z).$$

4. The zeroes of $\chi_{\Delta_A}(z)$ and $\chi_{\Delta_B}(z)$ are determined; they are precisely the elements of Δ_A and Δ_B respectively.

because all common factors cancel out. Although the degrees of $\chi_{S_A}(z)$ and $\chi_{S_B}(z)$ may be very large, the degrees of the numerator and denominator of the (reduced) rational function $\frac{\chi_{\Delta_A}(z)}{\chi_{\Delta_B}(z)}$ may be quite small. Thus, a relatively small number of sample points $(z_i, f(z_i))$ completely determine the rational function $f(z)$. Moreover, the size of $f(z)$ may be kept small and bounded by performing all arithmetic in an appropriately sized finite field.

The approach in [2] may thus be reduced conceptually to three fundamental steps, described in Protocol 1. This protocol assumes that an upper bound $\overline{m}$ on the number of differences m between two hosts is known *a priori* by both hosts. Section III-B describes an efficient, probabilistic, solution for the case when a tight bound $\overline{m}$ is not known.

A straightforward implementation of this algorithm requires expected computational time cubic in the size of the bound $\overline{m}$ and linear in the size of the sets S_A and S_B. However, in practice an efficient implementation can amortize much of the computational complexity. For example, hosts A and B can easily maintain their characteristic polynomial evaluations incrementally as data is added or deleted from their sets.

Overall, the algorithm in [2] communicates $\overline{m}$ computed samples from host A to B in order to reconcile at most $\overline{m}$ differences between the two sets; to complete the reconciliation, host B then sends back the $\overline{m}$ computed differences to A giving a total communication of $2\overline{m}$ integers. The only part of the communication complexity of Protocol 1 that depends on the set size is the representation of an integer, whose size is logarithmically related to the sizes of the sets being reconciled.

Thus, hosts A and B could each have millions of integers, but if the symmetric difference of their sets was at most ten then at most ten samples would have to be transmitted in each direction to perform reconciliation, rather than the millions of integers that would be transmitted in a trivial set transfer. Furthermore, this protocol does not require interactivity, meaning, for example, that host A could make his computed sample points available on the web; anyone else can then determine A's set simply by downloading these computed values, without requiring any computation from A. Example 1 demonstrates the protocol on two specific sets.

Example 1 A simple example of the interpolation-based synchronization protocol.

Consider the sets $S_A = \{1, 2, 4, 16, 21\}$ and $S_B = \{1, 2, 6, 21\}$ stored as 5-bit integers at hosts A and B respectively. We treat the members of S_A and S_B as members of a sufficiently large finite field (*i.e.* $\mathbb{F}_{71}$ in this case) so as to constrain the size of characteristic polynomial evaluations [2]. Assume an upper bound of $\overline{m} = 4$ on the size of the symmetric difference between S_A and S_B.

The characteristic polynomials for A and B are:

$$\chi_{S_A}(z) = (z-1)\cdot(z-2)\cdot(z-4)\cdot(z-16)\cdot(z-21),$$
$$\chi_{S_B}(z) = (z-1)\cdot(z-2)\cdot(z-6)\cdot(z-21).$$

The following table shows evaluation points, the corresponding characterstic polynomial values, and the ratio between the these values. All calculations are done over $\mathbb{F}_{71}$.

$z =$	-1	-2	-3	-4
$\chi_{S_A}(z)$	69	12	60	61
$\chi_{S_B}(z)$	1	7	60	45
$\chi_{S_A}(z)/\chi_{S_B}(z)$	69	22	1	55

Host B sends its evaluations to host A, who can now interpolate the following rational function from the evaluated sample points:

$$f(z) = \chi_{S_A}(z)/\chi_{S_B}(z) = \frac{z^2 + 51z + 64}{z + 65}$$

The zeros of the numerator and denominator are $\{4, 16\}$ and $\{6\}$ respectively, which are exactly equal to Δ_A and Δ_B.

Protocol 1 provides an efficient solution to the set reconciliation problem when the number of differences between two hosts (*i.e.* m) is known or tightly bounded. In many practical applications, however, a good bound is not known *a priori*. The following section describes a probabilistic technique for dealing with such cases.

B. Probabilistic scheme with an unknown upper bound

An information theoretic analysis [11] shows that if neither a distribution nor a non-trivial bound $\overline{m}$ is known on the differences between two host sets, then no deterministic scheme can do better than *slow sync*. However, with arbitrarily high probability, a probabilistic scheme can do much better. Specifically, the scheme in [2] suggests guessing such a bound $\overline{m}$ and subsequently verifying if the guess was correct. If the guessed value for $\overline{m}$ turns out to be wrong, then it can be improved iteratively until a correct value is reached.

Thus, in this case, we may use the following scheme to synchronize: First, hosts A and B guess an upper bound $\overline{m}$ and perform Protocol 1 with this bound, resulting in host A computing a rational function $\tilde{f}(z)$. If the function $\tilde{f}(z)$ corresponds to the differences between the two host sets, that is if

$$\tilde{f}(z) = \frac{\chi_{S_A}(z)}{\chi_{S_B}(z)}, \tag{2}$$

Example 2 An example of reconciliation when no bound $\overline{m}$ is known on the number of differences between two sets.

Consider using and incorrect bound $\overline{m} = 1$ in Example 1. In this case, host B receives the evaluation $\chi_{S_A}(-1) = 69$ from host A, and compares it to its own evaluation $\chi_{S_B}(-1) = 1$ to interpolate the polynomial

$$\tilde{f}(z) = \frac{z + 70}{1} \tag{3}$$

as a guess of the differences between the two hosts.

To check the validity of (3), host B then requests evaluations of A's polynomial at two random points, $r_0 = 38$ and $r_1 = 51$. Host A sends the corresponding values $\chi_{S_A}(r_0) = 23$ and $\chi_{r_1}(51) = 53$, which B divides by its own evaluations $\chi_{S_B}(r_0) = 38$ and $\chi_{S_B}(r_1) = 36$ to get the two verification points $f(r_0) = 66$ and $f(r_1) = 35$. Since the guessed function $\tilde{f}(z)$ in (3) does not agree at these two verification points, host B knows that the initial bound must have been incorrect. Host B may thus update its bound to $\overline{m} = 3$ and repeat the process.

then computing the zeroes of $f(z)$ will determine precisely the mutual difference between the two sets.

To check whether Equation (2) holds, host B chooses k random sample points r_i, and sends their evaluations $\chi_{S_B}(r_i)$ to host A, who uses these values to compute evaluations

$$f(r_i) = \frac{\chi_{S_A}(r_i)}{\chi_{S_B}(r_i)}.$$

By comparing $\tilde{f}(r_i)$ and $f(r_i)$, host A can assess whether Equation (2) has been satisfied. If the equation is not satisfied, then the procedure can be repeated with a different bound $\overline{m}$. Example 2 demonstrates this procedure.

In general, the two hosts keep guess $\overline{m}$ until the resulting polynomials agree in all k random sample points. A precise probabilistic analysis in [2] shows that such an agreement corresponds to a probability of error

$$\epsilon \le m \left[\frac{|S_A| + |S_B| - 1}{2^b} \right]^k. \tag{4}$$

Manipulating equation 4 and using the trivial upper bound $m \le |S_A| + |S_B|$, we see that one needs an agreement of

$$k \ge \left\lceil \log_\rho \left(\frac{\epsilon}{|S_A| + |S_B|} \right) \right\rceil \tag{5}$$

samples (where $\rho = \frac{|S_A|+|S_B|-1}{2^b}$) to get a probability of error at most ϵ for the whole protocol. Thus, for example, reconciling host sets of one million 64-bit integers with error probability $\epsilon = 10^{-20}$ would require agreement of $k = 2$ random samples.

We show in Section V-A that this verification protocol requires the transmission of at most $m + k$ samples and one random number seed (for generating random sample points) to reconcile two sets; the value k is determined by the desired probability of error ϵ according to Equation 5. Thus, though the verification protocol will require more rounds of communication for synchronization than the deterministic Protocol 1, it will not require transmission of significantly more bits of communication. We next show that the computational overhead of this probabilistic protocol is also not large.

IV. PDA Implementation

To demonstrate the practicality and effectiveness of our synchronization approach, we have implemented the CPIsync algorithm that was introduced in the previous sections on a real handheld device, that is, a Palm Pilot IIIxe Personal Digital Assistant.

Our program emulates the operation of a memo pad and provides a convenient testbed for evaluating the new synchronization protocol. Moreover, the successful implementation of this protocol on the computationally and communicationally limited Palm device suggests that the same can be done for more complicated, heterogeneous networks of many machines.

In this section, we describe our implementation and provide some experimental results for the specific case where the number of the differences, m, between the PDA and PC databases is either known or tightly bounded by $\overline{m}$ *a priori*. We show in Section IV-B how such a bound can be determined in many practical situations. In general, however, the tightness of the bound cannot be guaranteed, and it is much more efficient to employ the probabilistic scheme introduced in Section III-B. We describe an implementation of this more general scheme in Section V and show that its performance is close to the performance of a protocol that knows m *a priori*.

A. Experimental environment

Platform: Our experiments were performed on a Palm Pilot IIIxe with a 16-bit Motorola Dragonball processor and 8MB of RAM. The Palm was connected via a serial link to a Pentium III class machine with 512 MB of RAM.

Model: Our specific implementation of CPIsync emulates a memo pad application. As data is entered on the Palm, evaluations of the characteristic polynomial (described in Section III) are updated at designated sample points. Upon a request for synchronization, the Palm sends $\overline{m}$ of these evaluations to the desktop, corresponding to the presumed maximum number of differences between the data on the two machines. The desktop compares these evaluations to its own evaluations and determines the differences between the two machines, as described in Protocol 1. We compare CPIsync to an emulation of slow sync, which upon synchronization, sends all the Palm data to the desktop, and uses this information to determine the differences.

We do not address issues about which specific data to keep at the end of the synchronization cycle, but several techniques from the database literature may be adapted [12]. We also avoid issues of hashing by restricting entries to 15-bit integers. We note that, in practice, the hashing operation needs to be performed only once per entry, at the time that the entry is added to the data set; thus the complexity of hashing is not a bottleneck for synchronization. By restricting entries to 15 bits, we also avoid issues of multiple-precision arithmetic on the Palm, which can be easily solved by using the Chinese Remainder Theorem [13] to split arithmetic in large finite fields into arithmetic in several single-precision finite fields.

Finite field arithmetic is performed with Victor Shoup's Number Theory Library [14] and data is transferred in the Palm

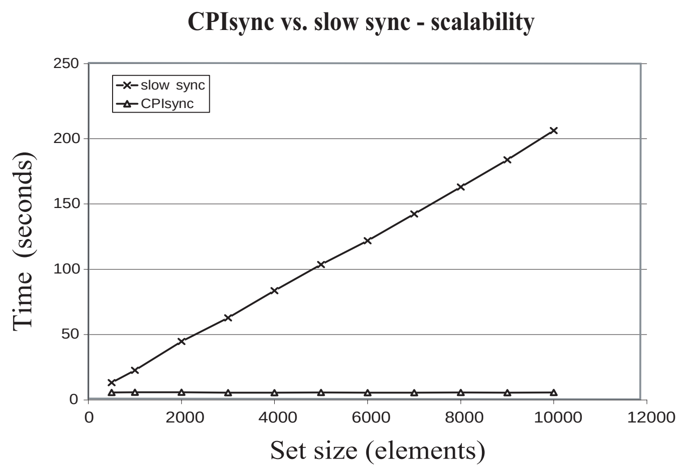


Fig. 6. A comparison of CPIsync and slow sync demonstrating the superiority of CPIsync for growing sets of data with a fixed number of differences (*i.e.* 101) between them.

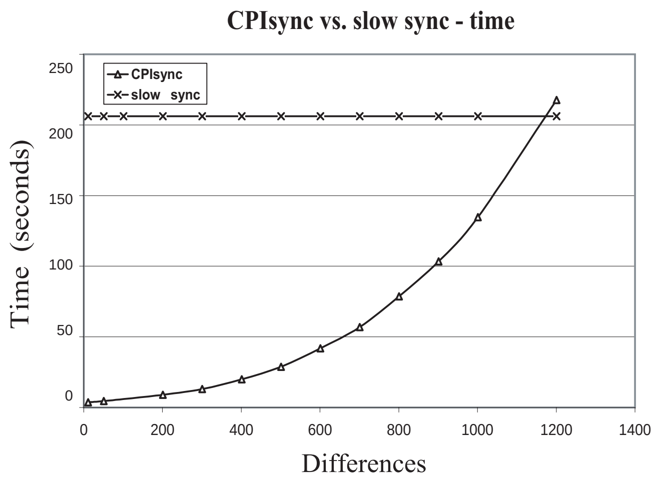


Fig. 7. A comparison of CPIsync and slow sync for sets having $10,000$ elements. The synchronization time is plotted as a function of the number of differences between the two sets.

Database File format. This data is converted to data readable by our Palm program using [15].

Metrics and Measurements: The two key metrics used in comparing CPIsync to slow sync are *communication* and *time*. Communication represents the number of bytes sent by each protocol over the link. For this metric, no experiments are needed as we have shown analytically that CPIsync will upload only $\overline{m}$ entries from the PDA, while slow sync will require the transfer of all the Palm entries. On the down link from the PC to the PDA, both protocols will transmit the same updates.

The time required for a synchronization to complete (*i.e.* the latency) is probably the most important metric from a user's point of view. For slow sync, the dominant component of the latency is the data transfer time, whereas for CPIsync the computation time generally dominates. Our experiments compare the latencies of CPIsync and slow sync in various scenarios. The synchronization latency is measured from the time at which the Palm begins to send its data to the PC until the time at which the PC determines all the differences between the databases. The results presented in the next section represent averages over 10 identical experiments.

Results: Figure 6 depicts the superior scalability of CPIsync over slow sync. In this figure, we have plotted the time used by each synchronization scheme as a function of data set size for a fixed number of differences between data sets.

It is clear from the resulting graphs that slow sync is markedly non scalable: the time taken by slow sync increases linearly with the size of the data sets. CPIsync, on the other hand, is almost independent of the data set sizes. Comparing Figure 4 to Figure 6 we observe that the qualitative behavior of CPIsync is similar to that of fast sync. The remarkable property of CPIsync is that it can be employed in any synchronization scenario, regardless of context, whereas fast sync is employed only when the previous synchronization took place between the same PC and the same PDA.

In Figure 7, we compare the performance of CPIsync to slow sync for data sets with fixed sizes but increasing number of differences. As expected, CPIsync performs significantly better than slow sync when the two reconciling sets do not differ by much. However, as the number of differences between the two sets grows, the computational complexity of CPIsync becomes significant. Thus, there exists a threshold where wholesale data transfer (*i.e.* slow sync) becomes a faster method of synchronization; this threshold is a function of the data set sizes as well as the number of differences between the two data sets. For the $10,000$ records depicted in the figure, this threshold corresponds to roughly $1,200$ differences.

By preparing graphs like Figure 7 for various different set sizes, we were able to produce a regression with a coefficient of determination [16] almost 1 that analytically models the performance of slow sync and CPIsync; the resulting threshold values are listed in Table I. Based on our theoretical development, the regression for slow sync is obtained by fitting the data to a linear function that depends only on the data set size, whereas for CPIsync the regression is obtained by fitting the data to a cubic polynomial that depends only on the number of differences. With such analytical models, we can determine a threshold for any given set size and number of differences between hosts, as illustrated by Figure 8.

Note that in a Palm PDA application like an address book or memo, changes between concurrent synchronizations typically involve only a small number of records. For such applications, CPIsync will usually be much faster than slow sync.

B. Determining an upper bound

The implementation of CPIsync described in the previous sections requires knowledge of a tight upper bound, $\overline{m}$, on the number of differing entries. One simple method for obtaining such a bound involves having both host A and host B count the number of modifications to their data sets since their last common synchronization. The next time that host A and host B synchronize, host A sends to host B a message containing its number of modifications, denoted $\overline{m}_A$. Host B computes its corresponding value $\overline{m}_B$ so as to form the upper bound $\overline{m} = \overline{m}_A + \overline{m}_B$ on the total number of differences between both hosts. Clearly, this bound $\overline{m}$ will be tight if the two hosts have performed mutually

Data set Size	*Differences*
250	175
500	253
1000	431
2500	620
3000	727
5000	899
10000	1177

TABLE I
THRESHOLD VALUES AT WHICH CPISYNC REQUIRES THE SAME AMOUNT OF TIME AS SLOW SYNC.

exclusive modifications. However, it may be completely off if the hosts have performed exactly the same modifications to their respective databases. This may happen if, prior to their own synchronization, both hosts A and B synchronized with a third host C, as in Figure 5. Another problem with this method is that it requires maintaining separate information for each host with which synchronization is performed; this may not be reasonable for larger networks. Thus, the simple method just described will be rather inefficient for some applications.

In the next section, we describe a probabilistic scheme that can determine a much tighter value for $\overline{m}$. This application is of fundamental importance because it allows CPIsync to achieve performance equivalent to fast sync in a general setting.

V. Practical Evaluation of the Probabilistic Scheme

The probabilistic scheme introduced in Section III-B can be implemented in various ways depending on the metric of interest. In this section, we propose two implementations based on the optimization of two different metrics.

A. Communication optimization

In one case, we may consider optimizing our implementation of the probabilistic scheme with respect to the amount of *communication* needed for reconciliation. It turns out that we can synchronize a PDA and a PC that differ in m entries by sending at most $m + k$ characteristic polynomial evaluations, where k is a small constant (see Section III-B).

Such a scheme can be implemented as follows: First the PDA sends to the PC evaluations of its own characteristic polynomial at a small number of pre-determined sample points and at k additional (different) random sample points. The former points are used to interpolate a rational function, corresponding to a guess of the differences between the two machines, and the latter points are used to verify the correctness of this guess. If the verification succeeds, then the synchronization is complete. On the other hand, if the verification fails, then the PC collects all the sample points seen so far into a guess of the differences between the two machines while at the same time requesting k additional random evaluations from the PDA to confirm this new guess. This procedure is iterated until verification succeeds, at which point synchronization is complete. Since m evaluations will necessarily be enough to completely determine up to

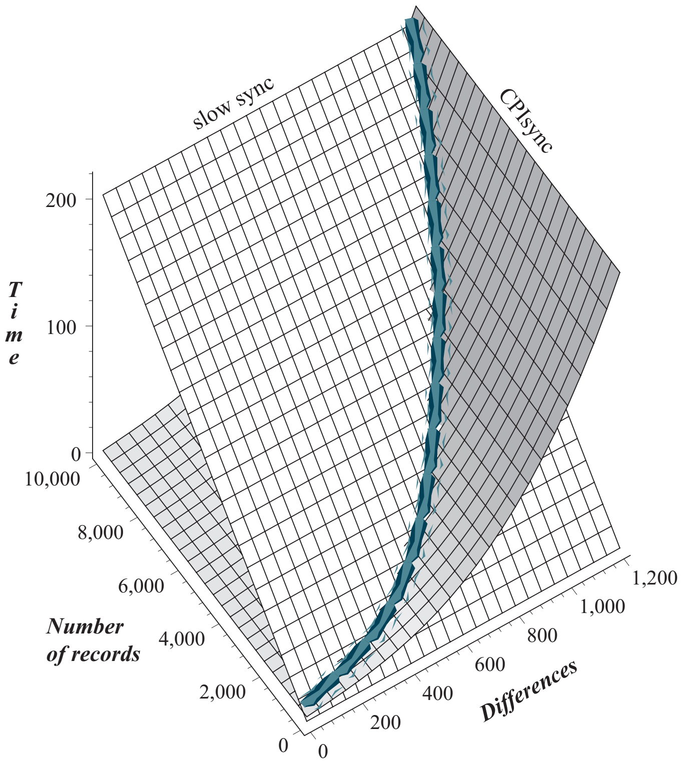


Fig. 8. A graph comparing slow sync and CPIsync for databases with varying numbers of records and with varying numbers of differences between databases. The patterned line depicts the threshold curve at which slow sync and CPIsync require the same amount of time to complete.

m differences, verification will necessarily succeed after at most $m + k$ transmissions.

B. Latency optimization

In a second case, we may consider optimizing our implementation for the purposes of minimizing the *latency* of our schemes (*i.e.* the overall time needed for synchronization). We thus propose a general probabilistic scheme whose completion time is at worst a constant α times larger than the time needed to synchronize two hosts when the number of differences between them is known *a priori*. This probabilistic scheme retains one of the essential properties of its deterministic counterpart: the synchronization latency depends chiefly on the number of differences between hosts. We prove that $\alpha = 4$ is an optimal bound for this scheme and show how to achieve it.

Our approach to this optimization relies in part on the data from Figure 7, reproduced in Figure 9. In the latter figure, we fit our data to a polynomial regression that interpolates the latency of CPIsync as a function of the number of differences m between two hosts. Since an exact value for m is not known at the start, the PDA and PC start with an initial guess $\overline{m}_1$ for an upper bound on m. In Figure 9, this initial guess corresponds to the value $\overline{m}_1 = 11$, which corresponds to a verification time of $t_1 = 3.65$ seconds. If verification fails for this guess, then we update our bound to the value $\overline{m}_2$ that corresponds to a verification time that is δ times larger than for $\overline{m}_1$ differences (*i.e.* $t_2 = \delta t_1$). In the case of Figure 9, we have chosen $\delta = 2$ giving $\overline{m}_2 = 151$ and $t_2 \approx 7.29$ seconds. At each iteration we guess

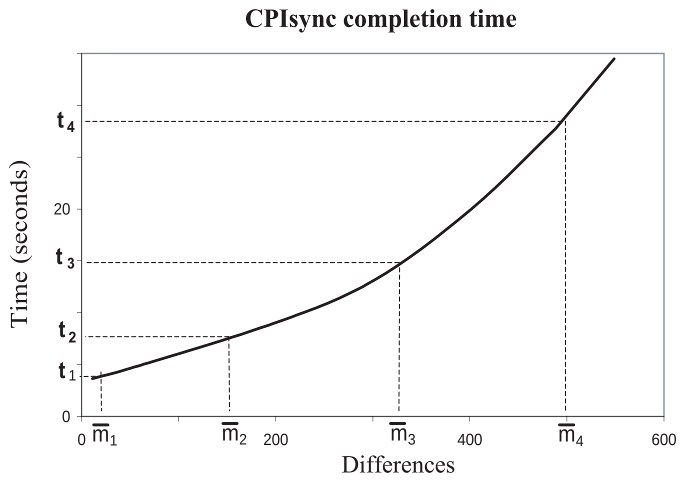


Fig. 9. A model of the approach used to optimize the latency of synchronization when no bound is known on the number of differences between data sets.

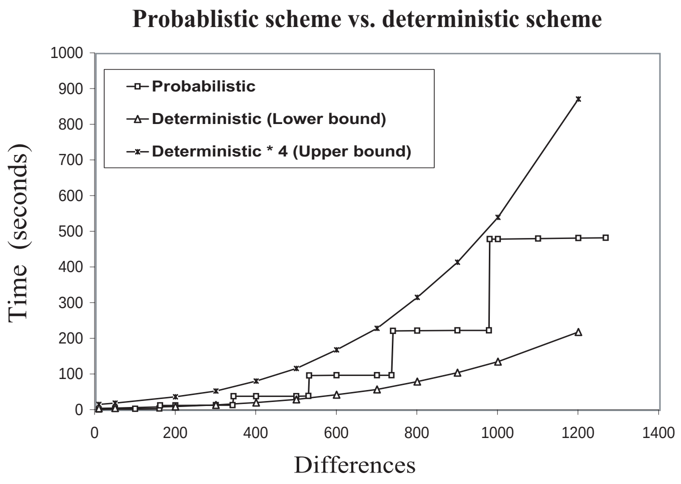


Fig. 10. A comparison of the probabilistic scheme with no known bound $\overline{m}$ to the deterministic scheme with a given value of m.

the bound $\overline{m}_i$ that corresponds to a verification time $t_i = \delta t_{i-1}$. We continue until verification succeeds for some guessed bound $\overline{m}_n$ requiring verification time $t_n = \delta^{n-1} t_1$.

Claim 1 *The latency-optimizing probabilistic scheme takes at most $\alpha(\delta) = \delta^2/(\delta - 1)$ times longer than a deterministic scheme with an* a priori *knowledge of the actual number of differences.*

Proof: Denote by $T^*(m)$ the synchronization latency when m is known *a priori*, and by $T(m)$ the synchronization latency required by this probabilistic scheme. Furthermore, denote by t_i the time needed for the i-th verification round in which $\overline{m}_i$ differences are guessed between the two hosts.

Suppose that a correct upper bound, $\overline{m}_n \geq m$, is obtained first at the n-th iteration, for $n > 1$. The total synchronization time required for the probabilistic scheme is then simply the sum of a geometric progression

$$T(m) = t_1 + \ldots + t_n = t_1 + \delta t_1 + \ldots + \delta^{n-1} t_1 = \frac{\delta^n - 1}{\delta - 1} t_1.$$

Note that $T^*(m) \geq t_{n-1} = \delta^{n-2} t_1$, since $\overline{m}_n$ is assumed to be the *first* correct upper bound m. We thus obtain

$$\frac{T(m)}{T^*(m)} \geq \frac{\delta^n - 1}{(\delta - 1)\delta^{n-2}}, \qquad \text{for all } n > 1. \tag{6}$$

It is easy to check that the right hand side of (6) is maximized when $n \to \infty$, meaning that $T/T^* \geq \delta^2/(\delta - 1)$. ∎

By examining the derivative of $\alpha(\delta)$ with respect to δ, one finds that this function attains a minimum value at $\delta = 2$, leading to an optimal ratio of $\alpha(2) = 4$. Thus, the best policy for this scheme is to double the verification time at each iteration.

Figure 10 illustrates the performance of this probabilistic scheme compared to that of the deterministic scheme. Note that the probabilistic results remain within the guaranteed factor 4 of the corresponding deterministic results.

VI. Related work

The general problem of data synchronization has been studied from different perspectives in the literature.

From a database perspective, the concept of disconnected operation, in which hosts can independently operate and subsequently synchronize, was established by the CODA file system [17]. The general model proposed in [17] is similar to the models used by several current mobile computing systems, including some PDAs.

The management of replicated, distributed, databases requires the development of sophisticated algorithms for guaranteeing data consistency and for resolving conflicting updates. Several architectures, such as BAYOU [18], ROAM [19], and DENO [20] have been proposed to address these important problems. We consider CPIsync to be complementary to these architectures. The CPIsync methodology permits the efficient determination of the differences between databases, while the mentioned architectures can be used to resolve which data to keep or to update once the differences are known.

The analysis of PDA synchronization protocols from the perspective of scalability of communications, as considered in this paper, is a relatively new area of research. The most closely related work we have found in the literature is the EDISON architecture proposed in [21]. This architecture relies on a centralized, shared server with which all hosts synchronize. The server maintains an incremental log of updates so that the hosts can always use fast sync instead of slow sync. Unlike CPIsync, this architecture is not designed for the general case where a device may synchronize with any other device on a peer-to-peer basis. In general, a distributed architecture based on peer-to-peer synchronization provides much better network performance, in terms of robustness and scalability, than a centralized architecture [18–20].

From an information-theoretic perspective, synchronization can also be modeled as a traditional error-correction problem. In this case, host B can be thought to have a corrupted copy of a database held by host A. When the corruptions are *non-destructive*, meaning that the corruptions only change data rather than adding new data or deleting old data, the problem of synchronizing the two databases is precisely the classical problem of error-correction [22]. Many sources [23–27] have addressed synchronization of such non-destructively corrupted databases. A more recent work [28] makes a direct link to coding theory by using a well-known class of good codes known as Reed-Solomon codes to affect such synchronizations.

However, the applications that we address in this work do not conform to this simplified synchronization model. It is generally not the case that database differences for mobile systems can be modeled as non-destructive corruptions. Instead, we need to allow for data to be added or deleted from anywhere within a database, as happens practically. Several sources [29, 30] have studied extended synchronization models in which the permitted corruptions include insertions, deletions, and modifications of database entries. Recently, Cormode, Paterson, Ṣahinhalp, and Vishkin [31] provided a probabilistic solution for such synchronization when a bound on the number of differences is known. However, all these algorithms assume a fundamental ordering of the host data. Thus, they synchronize not only the database contents, but also the order of the entries within each database. For example, a synchronization of the sets $\{1,3,5\}$ with $\{3,5,1\}$ would result in $\{1,3,5,1\}$ because order is considered significant.

In fact, many applications [5, 7–9, 32, 33] do not require both the synchronization of order and the synchronization of content, and the proposed synchronization technique takes advantage of this fact. For example, when synchronizing two address books, only the contact information for each entry needs to be communicated and not the location of the entry in the address book.

VII. Conclusion

In this paper, we have shown that the current performance of PDA synchronization schemes can be tremendously improved through the use of sophisticated computational methods [2, 3]. We have described, analyzed, and implemented a novel algorithm, termed CPIsync, for fast and efficient PDA synchronization. Our implementation demonstrated that it is possible to synchronize remote systems in a scalable manner, from the perspective of communication bandwidth and latency.

Specifically, we have shown that two hosts can deterministically reconcile their data in a real environment with a communication complexity depending only on the number of differences between the them, provided that they have good bound on this number of differences. We demonstrated the use of a probabilistic scheme for the cases where such a bound is not available. The accuracy of this probabilistic method can be made as good as desired, and its communication complexity is within an additive constant of the deterministic scheme that is supplied with the exact number of differences between both host sets.

Using analytical modeling, we also showed that the latency of this probabilistic scheme can be designed to be within a factor of 4 of the latency for the deterministic scheme. Thus, even without a knowing the number of differences between them, two hosts can reconcile with both a communication and latency that depends only on this number of differences. We presented experimental evidence of this phenomenon, demonstrating that, in most reasonable scenarios, CPIsync is substantially faster than the current reconciliation scheme implemented on the Palm PDA.

The CPIsync algorithm described in the paper is suitable not only for the specific application to PDAs, but also to any general class of problems where the difference in the data sets being reconciled is relatively small compared to the overall size of the data sets themselves. We believe that this scalable architecture will be essential in maintaining consistency in large networks.

Acknowledgments

The authors are grateful to Yaron Minsky for stimulating discussions and Felicia Trachtenberg for statistical advice.

References

[1] "Palm developer on-line documentation," http://palmos/dev/tech/docs.
[2] Y. Minsky, A. Trachtenberg, and R. Zippel, "Set reconciliation with nearly optimal communication complexity," Tech. Rep. TR1999-1778, TR2000-1796,TR2000-1813, Cornell University, 2000.
[3] Y. Minsky, A. Trachtenberg, and R. Zippel, "Set reconciliation with nearly optimal communication complexity," in *International Symposium on Information Theory*, June 2001, p. 232.
[4] "Frontline test equipment," http://www.fte.com.
[5] R.A. Golding, *Weak-Consistency Group Communication and Membership*, Ph.D. thesis, UC Santa Cruz, December 1992, Published as technical report UCSC-CRL-92-52.
[6] M. Harchol-Balter, T. Leighton, and D. Lewin, "Resource discovery in distributed networks," in *18th Annual ACM-SIGACT/SIGOPS Symposium on Principles of Distributed Computing*, Atlanta, GA, May 1999.
[7] R. van Renesse, "Captain cook: A scalable navigation service," In preparation.
[8] R. van Renesse, Y. Minsky, and M. Hayden, "A gossip-style failure detection service," in *Middleware '98: IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*, Nigel Davies, Kerry Raymond, and Jochen Seitz, Eds. 1998, pp. 55–70, Springer Verlag.
[9] K. Guo, M. Hayden, R. van Renesse, W. Vogels, and K. P. Birman, "GSGC: An efficient gossip-style garbage collection scheme for scalable reliable multicast," Tech. Rep., Cornell University, December 1997.
[10] M. Karpovsky, L. Levitin, and A. Trachtenberg, "Data verification and reconciliation with generalized error-control codes," *39th Annual Allerton Conference on Communication, Control, and Computing*, July 2001.
[11] M. Karpovsky, L. Levitin, and A. Trachtenberg, "Data verification and reconciliation with generalized error-control codes," *IEEE Trans. on Info. Theory*, 2001, submitted.
[12] A. Silberschatz, H.F. Korth, and S. Sudarshan, *Database System Concepts*, McGraw-Hill, third edition, 1999.
[13] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms*, MIT Press, 1990.
[14] V. Shoup, "NTL: A library for doing number theory," http://shoup.net/ntl/.
[15] "Pilot PRC-Tools," http://sourceforge.net/projects/prc-tools/.
[16] S. Weisberg, *Applied Linear Regression*, John Wiley and Sons, Inc., 1985.
[17] J. J. Kistler and M. Satyanarayanan, "Disconnected operation in the coda file system," *ACM Transactions on Computer Systems*, vol. 10, no. 1, pp. 3–25, 1992.
[18] D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser, "Managing update conflicts in bayou, a weakly connected replicated storage system," in *Proceedings of the 15th Symposium on Operating Systems Principles*, Copper Mountain Resort, Colorado, December 1995, ACM, number 22, pp. 172–183.
[19] D. Ratner, G. J. Popek P. Reiher, and R. Guy, "Peer replication with selective control," in *MDA '99, First International Conference on Mobile Data Access*, Hong Kong, December 1999.
[20] U. Cetintemel, P. J. Keleher, and M. Franklin, "Support for speculative update propagation and mobility in deno," in *The 22nd International Conference on Distributed Computing Systems*, 2001.
[21] M. Denny and C. Wells, "EDISON: Enhanced data interchange services over networks," May 2000, class project, UC Berkeley.
[22] F.J. MacWilliams and N.J.A. Sloane, *The Theory of Error-Correcting Codes*, North-Holland Publishing Company, New York, 1977.
[23] J.J. Metzner and E.J. Kapturowski, "A general decoding technique applicable to replicated file disagreement location and concatenated code decoding," *IEEE Transactions on Information Theory*, vol. 36, pp. 911–917, July 1990.

[24] D. Barbará, H. Garcia-Molina, and B. Feijoo, “Exploiting symmetries for low-cost comparison of file copies,” *Proceedings of the International Conference on Distributed Computing Systems*, pp. 471–479, 1988.
[25] D. Barbará and R.J. Lipton, “A class of randomized strategies for low-cost comparison of file copies,” *IEEE Transactions on Parallel Distributed Systems*, pp. 160–170, April 1991.
[26] W. Fuchs, K.L. Wu, and J.A. Abraham, “Low-cost comparison and diagnosis of large remotely located files,” *Proceedings of the Symposium on Reliability in Distributed Software and Database Systems*, pp. 67–73, 1996.
[27] J.J. Metzner, “A parity structure for large remotely located replicated data files,” *IEEE Transactions on Computers*, vol. C-32, no. 8, pp. 727–730, August 1983.
[28] K.A.S. Abdel-Ghaffar and A.E. Abbadi, “An optimal strategy for comparing file copies,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, no. 1, pp. 87–93, January 1994.
[29] T. Schwarz, R.W. Bowdidge, and W.A. Burkhard, “Low cost comparisons of file copies,” *Proceedings of the International Conference on Distributed Computing Systems*, pp. 196–202, 1990.
[30] Alon Orlitsky, “Interactive communication: Balanced distributions, correlated files, and average-case complexity.,” in *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science*, 1991, pp. 228–238.
[31] G. Cormode, M. Paterson, S.C. Ṣahinhalp, and U. Vishkin, “Communication complexity of document exchange,” *ACM-SIAM Symposium on Discrete Algorithms*, January 2000.
[32] R. Adams, “RFC1036: Standard for interchange of USENET messages,” December 1987.
[33] M. Hayden and K. Birman, “Probabilistic broadcast,” Tech. Rep., Cornell University, 1996.