Ef£cient PDA Synchronization

David Starobinski Ari Trachtenberg 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 inef£cient with respect to bandwidth usage, latency, and energy consumption, 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 and energy consumption of CPIsync is practical, and that the overall latency is typically much smaller than that of slow sync or alternative synchronization approaches based on Bloom £lters. Thus, CPIsync has potential for signi£cantly improving synchronization protocols for PDAs and, more generally, for heterogeneous networks of many machines.

Keywords: Personal Digital Assistant, mobile computing, data synchronization

A version of this paper appeared in

D. Starobinski, A. Trachtenberg and S. Agarwal, "Ef£cient PDA Synchronization", IEEE Transactions on Mobile Computing: **2**:1, January-March 2003.

An earlier version of this paper appeared in INFOCOM 2002.
The corresponding author is David Starobinski.
The authors are with the Department of Electrical and Computer Engineering at Boston University, Boston, MA. This work is supported in part by the National Science Foundation under NSF CAREER Grants ANI-0132802 and CCR-0133521.

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 of£ce or through a network. This feature plays an essential role in the vision of pervasive computing, in which any mobile device would ultimately be able to access and synchronize with any networked data.

Current PDA synchronization architectures, though simple, are often inef£cient. With some 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 inef£cient 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 consider the application of 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[1] 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 the 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.

[1]We use the generic term *hosts* to refer to either a PC or PDA.

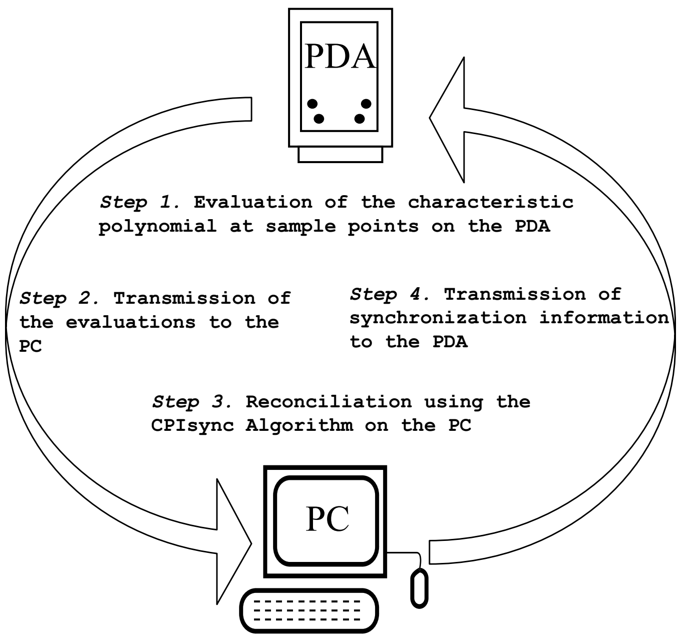
PDA
Step 1. Evaluation of the characteristic polynomial at sample points on the PDA
Step 2. Transmission of the evaluations to the PC
Step 4. Transmission of synchronization information to the PDA
Step 3. Reconciliation using the CPIsync Algorithm on the PC
PC

Fig. 1. The overall mode of operation of the CPIsync algorithm.

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 Fig. 1.

We have implemented CPIsync on a Palm Pilot IIIxe, a popular and representative PDA. Our experimental results show that CPIsync can perform signi£cantly better (sometimes, by order of magnitudes) than slow sync and alternative synchronization approaches based on Bloom £lters [4, 5] in terms of latency and bandwidth usage. Moreover, our experimental evidence shows that these savings translate to a corresponding ef£ciency of energy consumption. On the other hand, as the number of differences between hosts increase, the computational complexity of CPIsync becomes signi£cant. Thus, if two hosts differ signi£cantly, then Bloom £lters or even wholesale data transfer may become faster methods of synchronization. The threshold number of differences at which CPIsync loses its advantage over other protocols 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, thus, propose to make use and implement a probabilistic technique from [2] for testing the correctness of a guessed upper bound. If one guess turns out to be incorrect, then the guess can be modi£ed in a second attempted synchronization, and so forth. The error of this probabilistic technique can be made arbitrarily small. We 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.

A. Organization

In the next section we begin with a review of the synchronization techniques currently used on the Palm OS computing platform. We indicate the limitations of these techniques and of several seemingly reasonable alternatives. This section also describes other related work in the literature. 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 speci£c implementation of CPIsync on a Palm Pilot IIIxe. We also present experimental results on bandwidth, latency, and energy usage for the case where a tight bound on the number of differences is known *a priori*, and provide comparisons to implementations of slow sync and Bloom £lters. 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*. Finally, we provide our conclusions in Section VI.

II. Background

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. This implementation was clearly designed for a very narrow usage model, and is not scalable with network size or data storage. We thus also provide seemingly reasonable alternatives to the Palm's synchronization protocol and discuss other related work, before presenting, in Section III, our chosen solution based on characteristic polynomial interpolation.

A. The Palm Synchronization Protocol

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 identi£er, a pointer to the record’s memory location, and status ¤ags. The status ¤ags remain clear only if the data record has not been modi£ed since the last synchronization event. Otherwise the status ¤ags indicate the new status of the record (*i.e.* modi£ed, 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 ¤ags 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 ¤ags 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 ¤ags 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 Fig. 2.

The slow sync mode, which amounts to a wholesale transfer, is signi£cantly less ef£cient than fast sync. In particular, the latency, amount of communication, and energy required by slow sync increase linearly with the number of records stored in the device, independently of the actual

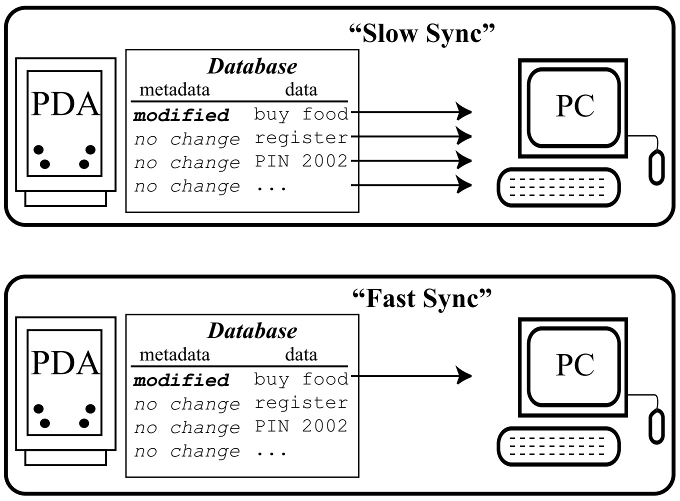

Fig. 2. The two modes of the Palm HotSync protocol. In the "slow sync" mode all the data is transferred. In the "fast sync" mode only modi£ed entries are transferred between the two databases.

number of differing records. 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 [6].

B. Timestamps

Another plausible synchronization solution is to use timestamps or version control to aid in discovering what data elements a given host is missing. When only two hosts are synchronizing, as happens if a PDA is synchronized always with just one home computer, then timestamps can and do work very effectively. In these cases, synchronization is accomplished by a simple exchange all items modi£ed since the previous synchronization, as done with the fast sync operation mode.

Unfortunately, the case is much more complicated when synchronizing more than two machines (*e.g.* a PDA with a home machine and a work machine). In this case, the use of timestamps can result in an inef£cient communication cost for synchronization, as depicted in Fig. 3. In addition, in a network scenario such as envisioned by the SyncML [7] initiative, 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

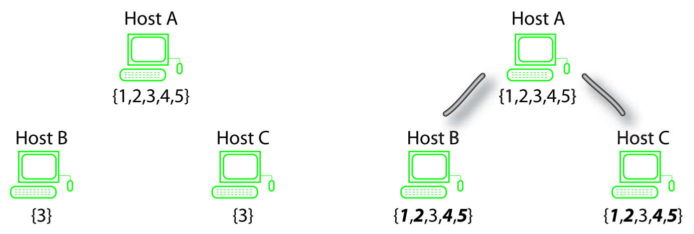

Fig. 3. Ineff£ciency of timestamps. Hosts B and C synchronize when each has item 3 (left £gure). Thereafter hosts B and C each synchronize with host A, noting the addition of items $1, 2, 4$, and 5. When hosts B and C then synchronize, modi£cation records require transmission of eight differences, marked in bold, whereas, in fact, there are none.

unpredictably.

C. Bloom £lter

Another possible technique for synchronization is the Bloom £lter [4, 5], which hashes each data element into several 1 bits in an array. This array, known as the Bloom £lter of the set, can represent the set very concisely. To test whether a speci£c element x is in a set, one need only check whether the appropriate bits are 1 in the Bloom £lter of the set; if they are not, then x is certainly not in the set, but otherwise nothing can be said about the existence of x in the set. In the latter case, it is possible for two hosts to incorrectly synchronize by surmising that x is on both hosts (*i.e.* a false positive indication for one of the hosts) when, in fact, it is only on one host.

This probability of a false positive, p_f, depends on the length of the Bloom £lter, l, the number of elements in the original data set, $|S|$, and the number of hash functions, h, used to compute the Bloom £lter. The false positive probability is given asymptotically (*i.e.* for large values of l) by [5]:

$$p_f = (1 - e^{-h|S|/l})^h. \tag{1}$$

We can rewrite (1) as

$$\frac{l}{|S|} = \frac{-h}{ln\left(1 - \sqrt[h]{p_f}\right)} \tag{2}$$

$$\geq \frac{1}{ln(2)} \qquad \text{for } p_f \leq \tfrac{1}{2}, \tag{3}$$

where the inequality (3) can be determined from inspecting the derivative of (2). As such, in order to maintain a constant false positive probability, the length of the Bloom £lter has to be increased linearly with the size of the data set.

The linear length of Bloom £lters (with respect to the set size) and their potentially high false positive rate make them less attractive for some practical set reconciliation problems, as we shall see in Section IV-C. In fact, almost all existing data compression schemes, whether lossless or lossy, compress data by at most a constant fraction, and will end up with a compressed signature whose length is linear in the original set size.

D. *Other 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 £le system [8]. The general model proposed in [8] 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 con¤icting updates. Several architectures, such as BAYOU [9], ROAM [10], and DENO [11] have been proposed to address these important problems. We consider CPIsync to be complementary to these architectures. The CPIsync methodology permits the ef£cient 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 [12]. 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 [9–11]. Recent work in [6] highlights the trade-offs between such considerations in the context of mobile device synchronization.

From an information-theoretic perspective, synchronization can also be modeled as a traditional error-correction problem. In the case where synchronizing hosts differ by in-place corruptions (*i.e.* rather than insertions or deletions), error-correcting codes can be used to "correct" differences [13]. However, the results in [14] show that any classical error-correcting code can also be used for set reconciliation. Reed-Solomon codes (and their more general form of BCH codes) provide one particularly good application because, over a £xed £nite £eld, their decoding time depends only on the number of errors in a transmitted vector. Transferred to the parlance of set reconciliation, this means that the decoding time depends mostly on the number of differences between sets (rather than on the overall sizes of the sets). The solution provided by Reed-Solomon codes is related to CPIsync [15] and provides a good means of data synchronization.

III. Characteristic Polynomial Interpolation-Based Synchronization

The key challenge to ef£cient 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 this section, we present a family of such protocols based on characteristic polynomial interpolation-based synchronization (CPIsync).

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 elements of the sets are important and not their relative positions within the set. Note also that the integer sets being synchronized can generically encode all types of data. In [2, 3] this formalization is called the *set reconciliation* problem. Natural examples of set reconciliation include synchronization of bibliographic data [16], resource availability [17, 18], data within gossip protocols [19, 20], or memos and address books. On the other hand, synchronization of edited text is not a clear-cut example of set reconciliation because the structure of data in a £le encodes information; for example, a £le containing the string "a b c" is not the same as a £le containing the string "c b a".

The set reconciliation problem is intimately linked to design questions in coding theory and graph theory [14, 21] 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 speci£cally for ef£cient reconciliation. To this end, [2] makes use of a characteristic polynomial $\chi_S(Z)$ of a set $S = \{x_1, x_2, \ldots, x_n\}$, de£ned to be:

$$\chi_S(Z) = (Z - x_1)(Z - x_2)(Z - x_3)\cdots(Z - x_n). \tag{4}$$

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

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

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 £nite £eld.

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-C describes an ef£cient, 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 ef£cient implementation can amortize much of the computational complexity over insertions and deletions into the sets, thus maintaining characteristic polynomial evaluations incrementally.

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 coef£cients 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.

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. By contrast, the information theoretic minimum amount of communication needed for reconciling sets of b-bit integers with an intersection of size N is [2]:

$$C \geq \lg \left[\binom{2^b - N - \overline{m}}{\overline{m}} \right] \approx b\overline{m} - \overline{m} \lg \overline{m} \text{ bits},$$

which corresponds to sending roughly $\overline{m} - \frac{\lg \overline{m}}{b}$ integers. One should also note that the only part of the communication complexity of Protocol 1 that depends on the set size is the representation of an integer.

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 speci£c 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 suf£ciently large £nite £eld (*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 characteristic 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.

B. Determining an upper bound

The CPIsync protocol described in the previous section requires knowledge of a tight upper bound, $\overline{m}$, on the number of differing entries. One simple method for possibly obtaining such a bound involves having both host A and host B count the number of modi£cations 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 modi£cations, 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 exclusive modi£cations. However, it may be completely off if the hosts have performed exactly the same modi£cations 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 Fig. 3. 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 large networks. Thus, the simple method just described could be rather inef£cient for some applications.

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

C. Probabilistic scheme with an unknown upper bound

In the general setting where no knowledge on a upper bound is provided, it is impossible to reconcile sets to a theoretical certainty without performing the equivalence of a slow sync [15, 22]. Fortunately, a probabilistic scheme can synchronize, with arbitrarily low probability of error, much more ef£ciently than the deterministic optimum given by *slow sync*. Speci£cally, 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

$$\tilde{f}(z) = \frac{\chi_{S_A}(z)}{\chi_{S_B}(z)}, \tag{5}$$

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

To check whether Equation (5) holds, host B chooses k random sample points r_i, and sends their

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

Consider using an 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

$$\tilde{f}(z) = \frac{z + 70}{1} \tag{6}$$

as a guess of the differences between the two hosts.

To check the validity of (6), 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 $\chi_{S_A}(r_1) = 53$, which B divides by its own evaluations $\chi_{S_B}(r_0) = 38$ and $\chi_{S_B}(r_1) = 36$ to get the two veri£cation points $f(r_0) = 66$ and $f(r_1) = 35$. Since the guessed function $\tilde{f}(z)$ in (6) does not agree at these two veri£cation 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.

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 (5) has been satis£ed. If the equation is not satis£ed, then the procedure can be repeated with a different bound $\overline{m}$. Example 2 demonstrates this procedure.

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

$$\epsilon \leq m\left[\frac{|S_A| + |S_B| - 1}{2^b}\right]^k, \tag{7}$$

assuming that each integer has a b-bit binary representation. Manipulating Equation (7) and using the trivial upper bound $m \leq |S_A| + |S_B|$, we see that one needs an agreement of

$$k \geq \left\lceil \log_\rho\left(\frac{\epsilon}{|S_A| + |S_B|}\right)\right\rceil \tag{8}$$

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 veri£cation 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 (8). Thus, though the veri£cation protocol will require more rounds of communication for synchronization than the deterministic Protocol 1, it will not require transmission of signi£cantly more bits of communication. We show in Section V 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 speci£c 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*. In general, however, the tightness of the bound cannot be guaranteed, and it is much more ef£cient to employ the probabilistic scheme described in Section III-C. We detail 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 speci£c 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. In Section IV-B, 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 also compare CPIsync to a Bloom £lter implementation in Section IV-C. To enable a fair comparison, our implementation made use of perfect XOR hashes, chosen for their very simple implementation on the computationally limited PDA. In addition, all hash computations for set elements were computed and stored off-line. A Bloom £lter synchronization run is thus executed as follows:

- The PC and PDA exchange Bloom £lters.
- The PDA uses the PC's Bloom £lter and the hash values of its own data set to calculate all the data elements the PC is missing. The PDA then sends these missing elements to the PC.
- Likewise, the PC uses the PDA's Bloom £lter to calculate and transmit all the data elements the PDA is missing.

We do not address issues about which speci£c data to keep at the end of the synchronization cycle, but several techniques from the database literature may be adapted [23]. 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. Nevertheless, we restrict entries to 15 bits so as to avoid issues of multiple-precision arithmetic on the Palm, which can be otherwise ef£ciently implemented using well-known techniques [24].

Finite £eld arithmetic is performed with Victor Shoup's Number Theory Library [25] and data is transferred in the Palm Database File (PDB) format. This data is converted to data readable by our Palm program using PRC-tools [26]. It is to be noted that the PDB format stores data as raw

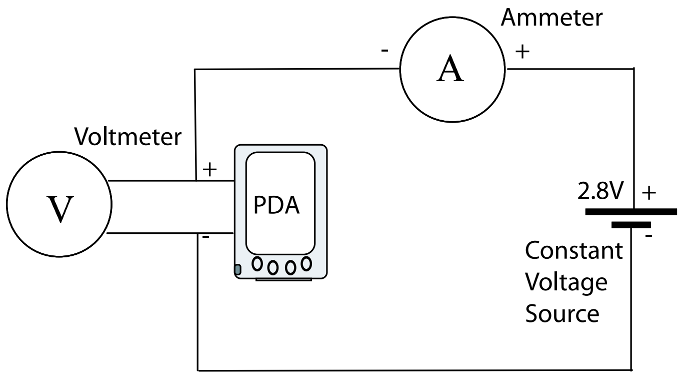

Fig. 4. A simple circuit for measuring PDA energy usage.

text, so that a 5-digit integer is actually stored as 5 one-byte long characters presumably to avoid the computational expense of packing and unpacking data. We maintain this format in our testing to insure the fairness of our comparisons, and thus store 15-bit numbers in 5 bytes.

Metrics and Measurements: The three major metrics used in comparing CPIsync to other approaches are communication, time and energy.

Communication represents the number of bytes sent by each protocol over the link. For this metric, 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. The communication complexity of Bloom £lters is linearly related to the data set size as well, as shown by Equation (2). On the down link from the PC to the PDA, all three 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 and Bloom £lters the computation time generally dominates. Our experiments compute and compare the latencies of CPIsync, slow sync and Bloom £lters 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 sections represent averages over 10 identical experiments.

The third metric of interest is *energy*. Minimizing the amount of energy consumed during synchronization events is of paramount importance for extending the battery life-time of mobile devices. The total energy expended during synchronization can be categorized as follows:

- CPU energy: energy associated with processing costs, such as computing and writing to memory the characteristic polynomial evaluations in CPIsync, and
- Communication energy: energy associated with communications between devices, including processing costs for encoding and decoding the transmitted data.

Ideally, energy consumption can be measured by integrating the product of instantaneous current $i(t)$ and voltage $v(t)$ values over time, *i.e.* , $E = \int_0^T i(t)v(t)dt$, where T represents the total synchronization time. Fortunately, experimental data shows that the Palm Pilot generally draws a constant current from a constant voltage source during different synchronization operations. For instance, the current intensity is constantly about 69 mA during data transmissions, 65 mA during characteristic polynomial evaluations and 14 mA in idle state. As such, we were able to determine energy usage fairly accurately using the setup in Fig. 4, by simply multiplying £xed current and voltage values during each synchronization operation by the time taken by the operation.

B. Comparison with Slow Sync

Fig. 5 depicts the superior scalability of CPIsync over slow sync. In this £gure, we have plotted the time used by each synchronization scheme as a function of data set size for a £xed 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. 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 Fig. 6, we compare the performance of CPIsync to slow sync for data sets with £xed sizes but increasing number of differences. As expected, CPIsync performs signi£cantly 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 signi£cant. Thus,

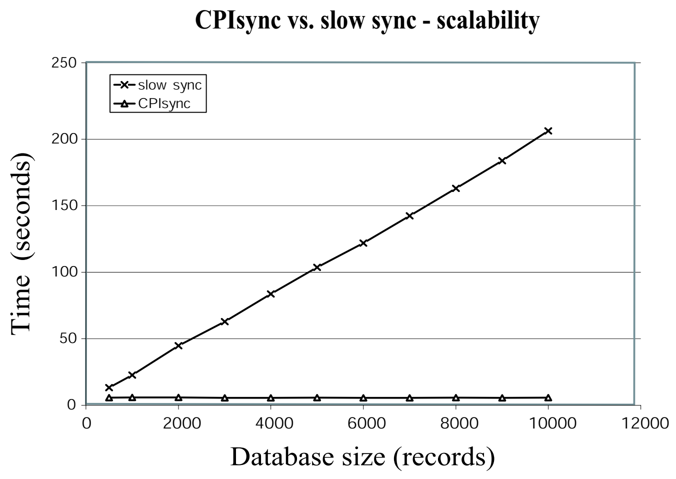

Fig. 5. A comparison of CPIsync and slow sync demonstrating the superiority of CPIsync for growing sets of data with a £xed number of differences (*i.e.* 101) between them.

Data set Size	*Differences*
250	175
500	253
1,000	431
2,500	620
3,000	727
5,000	899
10,000	1,177

TABLE I

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

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 £gure, this threshold corresponds to roughly $1,200$ differences.

By preparing graphs like Fig. 6 for various different set sizes, we were able to produce a regression with a coef£cient of determination [27] almost 1 that analytically models the performance

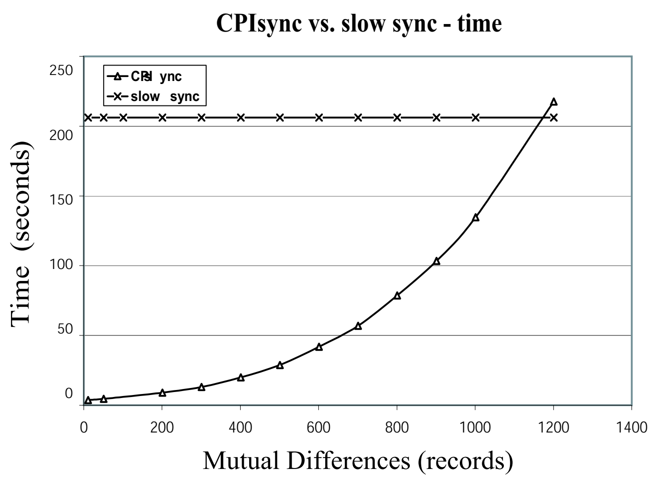

Fig. 6. A comparison of CPIsync and slow sync for sets having $10,000$ elements. The synchronization time is plotted as a function of the number of differences between the two sets.

of slow sync and CPIsync; the resulting threshold values are listed in Table I. The regression for slow sync is obtained by £tting the data to a linear function that depends only on the data set size, whereas for CPIsync the regression is obtained by £tting the data to a cubic polynomial that depends only on the number of differences. With such analytical models, one can determine a threshold for any given set size and number of differences between hosts [28]. In a Palm PDA application like an address book or memo, changes between concurrent synchronizations typically involve only a small number of records, in which case CPIsync will usually be much faster than slow sync.

Predictably, slow sync also expends much more energy to synchronize than CPIsync, growing linearly with database size as seen in Fig. 7. In particular, for a database size of $10,000$ records, and 100 differences, slow sync will consume about 10 mAH of energy, close to twenty times more than CPIsync. Considering that the typical amount of energy stored on a new battery is about $1,000$ mAH, we observe that CPIsync can provide signi£cant energy savings with respect to slow sync.

On the other hand, CPIsync does have a one-time set-up cost that depends on the database size, as shown in Fig. 8. This cost is due to the computation of the characteristic polynomial evaluations, when the entire database is uploaded to the PDA for the £rst time. Note that for databases smaller than $10,000$ records, the cost of this operation is still smaller than that of one slow sync.

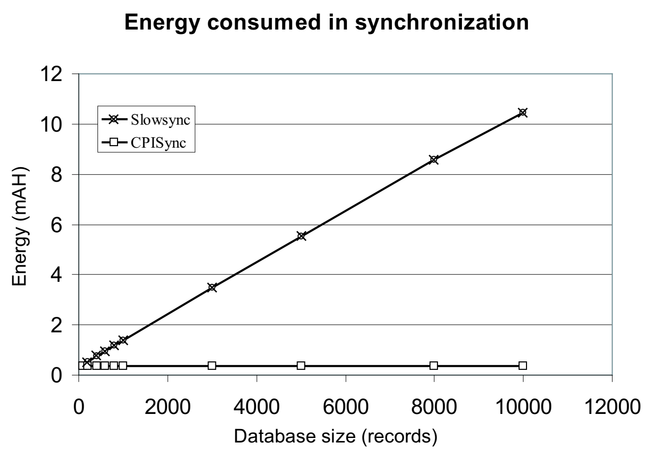

Fig. 7. A comparison of the energy consumed during synchronization using CPIsync and slow sync, for a £xed number of differences.

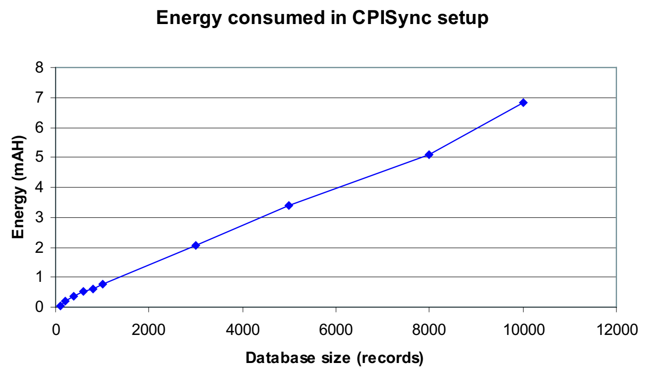

Fig. 8. Energy expended during the one-time setup of CPIsync, including calculation of characteristic polynomial evaluations.

C. Comparison with Bloom Filters

In this section, we compare the performance of CPIsync and that of synchronizing with a 4KB-long Bloom £lter, as described in Section II-C. The number of hash functions, h, is set to 17 as a reasonable compromise between computational ef£ciency, which dictates h be as small as possible, and the optimal number of hashes (*i.e.* $h = 23$) needed to minimize the probability of false positives.

The synchronization latencies of CPIsync and Bloom £lters are depicted in Fig. 9 for the case of two machines with $1,000$ records each, and an increasing number of differences. We note that Bloom £lters have a £xed latency component that is independent of the number of differences. This latency component is mostly due to computations performed by the PDA over the PC's Bloom

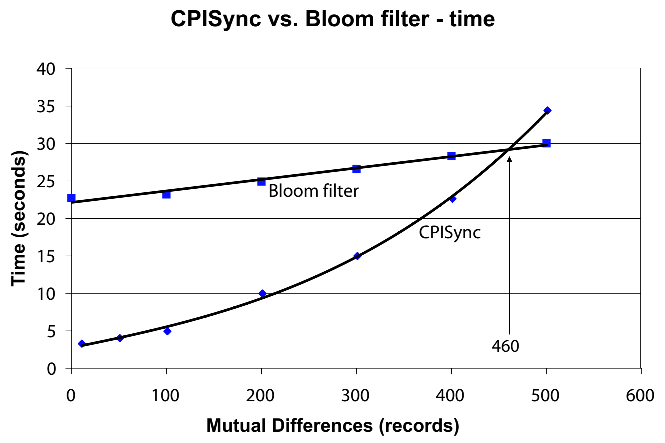

Fig. 9. Comparison of the synchronization latency between Bloom £lters and CPIsync for $1,000$ records.

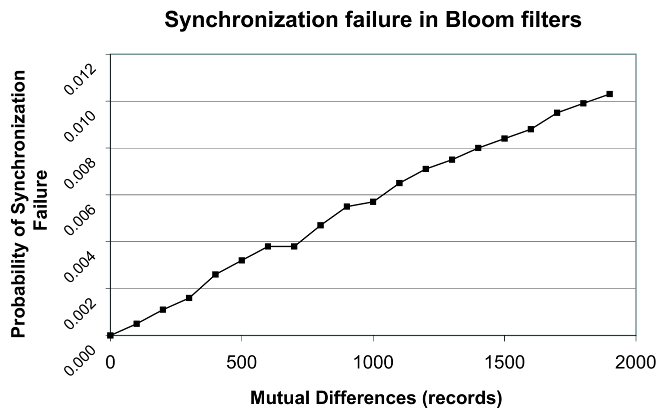

Fig. 10. The probability with which Bloom £lter synchronization will fail to completely synchronize two $1,000$-record databases with increasing numbers of differences.

£lter, in order to £nd out which of the PDA's elements the PC is missing. This computation time grows linearly with the size of the database, as the presence or absence of each element needs to be veri£ed.

As in the case of slow sync, we observe the existence of a threshold below which CPIsync provides for a smaller synchronization latency and above which Bloom £lters perform better. In Fig. 9, this threshold corresponds to about 460 differences. It is high enough for CPIsync to perform better than Bloom £lters in the majority of practical PDA synchronization cases.

As discussed in Section II-C, an important practical limitation of Bloom £lters is the possibility of a synchronization failure due to false positives. The problem with a false positive is that the

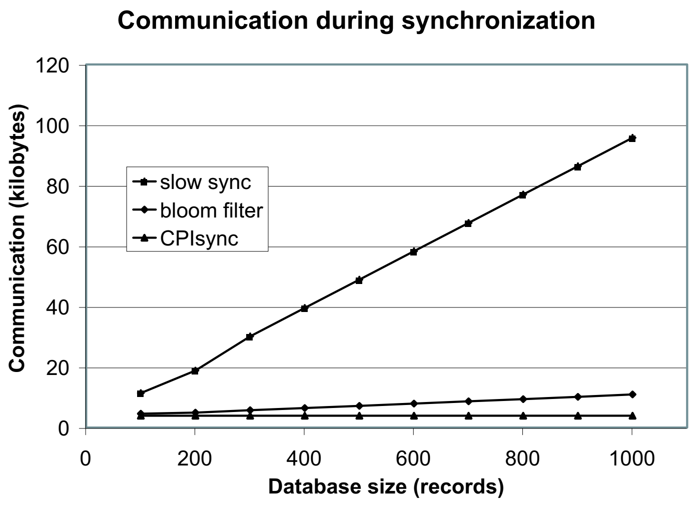

Fig. 11. Comparison between the communication complexities (*i.e.* number of bytes transmitted) of slow sync, Bloom £lter and CPIsync, as a function of the data set size, for 20 differing entries.

PDA mistakenly believes that the PC has a certain data element because the hash function values corresponding to that data element are otherwise present in PC's Bloom £lter (or vice-versa). False positives, consequently, lead to synchronization failures, in which the PDA and the PC end up with different databases, a highly undesirable situation for most users.

Fig. 10 depicts the probability of a synchronization failure, as a function of the number of differences between the PC's and PDA's databases. As expected, this probability increases with the number of differences, since only differing data elements may lead to false positives. We observe that with $2,000$ differences, the probability of a synchronization failure is about 0.01, which is non-negligible. This probability could be reduced by increasing the size of the Bloom £lter, but this would come at additional communication and computation cost.

We note that Bloom £lters may also have quite large memory requirements due to the need of storing hash evaluations. In our implementation, supporting memory for Bloom synchronization took up 384KB of memory for a $1,000$ record database on the PDA, whereas CPIsync required only 26KB of metadata storage.

The communication complexities of slow sync, CPIsync and Bloom £lters are compared in Fig. 11, for varying database sizes and a £xed number of differences (*i.e.* 20). For each given database size, $|S|$, the size of the Bloom £lter, l, was obtained from (2) using the values $p_f = 10^{-7}$

for the probability of a false positive and $h = 17$ for the number of hashes. As expected, Fig. 11 shows that the communication complexity of CPIsync does not depend on the database size. Although Bloom £lters synchronization achieves a signi£cant compression gain compared to slow sync, its communication complexity still grows linearly with the database size. As an example, for 1000 records on both the PC and the PDA, Bloom £lters require the transfer of over 11KB, while CPIsync requires slightly less than 4KB of communication.

V. Practical Evaluation of the Probabilistic Scheme

The probabilistic scheme introduced in Section III-C 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: communication and latency.

A. Communication optimization

We may consider optimizing our implementation of the probabilistic scheme with respect to the amount of *communication* needed for synchronization. In this case, 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 that determines the probability of error of the scheme (see Section III-C).

Such a scheme can be implemented as follows: £rst, 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 pseudo-random sample points (chosen according to a shared, or transmitted, seed). 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 veri£cation succeeds, then the synchronization is complete. On the other hand, if the veri£cation 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 pseudo-random evaluations from the PDA to con£rm this new guess. This procedure is iterated until veri£cation succeeds, at which point synchronization is complete. Since m evaluations will necessarily be enough to completely determine up to m differences, veri£cation will necessarily succeed after at most $m + k$ evaluations.

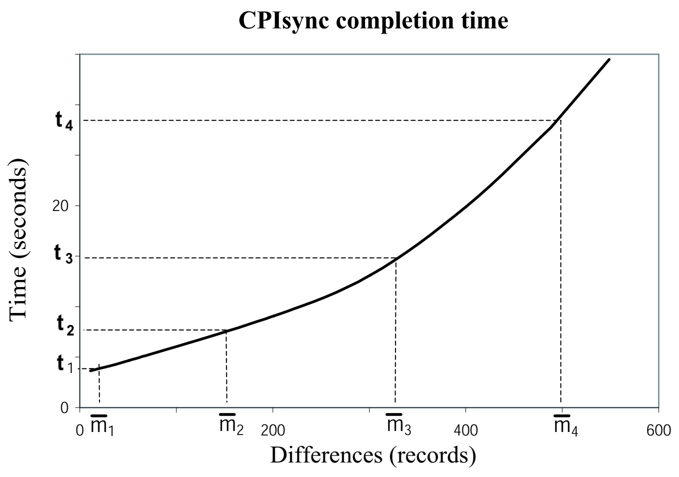

Fig. 12. A model of the approach used to optimize the latency of synchronization when no bound is known on the number of differences between data sets.

B. Latency optimization

We may alternatively consider optimizing the *latency* of our schemes (*i.e.* the overall time needed for synchronization). In this section we show that a simple modi£cation of the scheme in Section III-C will provide 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 modi£ed scheme retains one of the essential properties of its deterministic counterpart: the synchronization latency depends chie¤y 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 is based, in part, on the data from Fig. 6, reproduced in Fig. 12. In the latter £gure, we £t 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 Fig. 12, this initial guess corresponds to the value $\overline{m}_1 = 11$, which corresponds to a veri£cation time of $t_1 = 3.65$ seconds. If veri£cation fails for this guess, then we update our bound to the value $\overline{m}_2$ that corresponds to a veri£cation time that is δ times larger than for $\overline{m}_1$ differences (*i.e.* $t_2 = \delta t_1$). In the case of Fig. 12, we have chosen $\delta = 2$ giving $\overline{m}_2 = 151$ and $t_2 \approx 7.29$ seconds. At each iteration we guess the bound $\overline{m}_i$ that corresponds to a veri£cation time $t_i = \delta t_{i-1}$. We continue until veri£cation succeeds for some guessed bound $\overline{m}_n$ requiring veri£cation 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 veri£cation 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 £rst 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 *£rst* correct upper bound m. We thus obtain

$$\frac{T(m)}{T^*(m)} \geq \frac{\delta^n - 1}{(\delta-1)\delta^{n-2}}, \qquad \text{for all } n > 1. \tag{9}$$

It is easy to check that the right hand side of (9) 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 £nds that this function attains a minimum value at $\delta = 2$, leading to an optimal worst-case ratio of $\alpha(2) = 4$. Thus, the best policy for this scheme is to double the veri£cation time at each iteration.

Fig. 13 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. Conclusion

In this paper, we have shown that the current performance of PDA synchronization schemes can be tremendously improved through the use of more sophisticated existing computational meth-

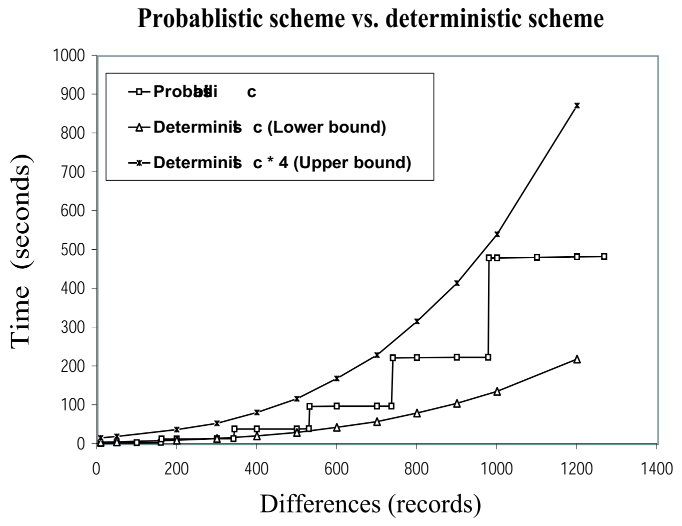

Fig. 13. A comparison of the probabilistic scheme with no known bound $\overline{m}$ to the deterministic scheme with a given value of m.

ods [2, 3]. We have described and analyzed a novel implementation, termed CPIsync, for fast and ef£cient PDA synchronization. Our implementation has demonstrated that it is possible to synchronize remote systems in a scalable manner, from the perspective of communication bandwidth, latency, and energy use.

Using CPIsync, two hosts can deterministically reconcile their data in a real environment with a communication complexity essentially depending only on the number of differences between the them, provided that they have a good bound on this number of differences. We have also demonstrated the use of a probabilistic scheme for the cases where such a bound is not available. This scheme can have an arbitrarily small probability of error, 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 have also shown 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 and energy-ef£cient than the current reconciliation scheme implemented on the Palm PDA. Moreover, CPIsync can outperform other fast synchronization approaches, such as

Bloom £lters, for which communication and latency were shown to depend linearly on the data sets' size.

The CPIsync algorithm described in the paper is suitable not only for the speci£c 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] Burton H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, July 1970.

[5] Li Fan, Pei Cao, Jussara Almeida, and Andrei Z. Broder, "Summary cache: a scalable wide-area Web cache sharing protocol," *IEEE/ACM Transactions on Networking*, vol. 8, no. 3, pp. 281–293, 2000.

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

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

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

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

[10] 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.

[11] 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.

[12] M. Denny and C. Wells, "EDISON: Enhanced data interchange services over networks," May 2000, class project, UC Berkeley.

[13] K.A.S. Abdel-Ghaffar and A.E. Abbadi, "An optimal strategy for comparing £le copies," *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, no. 1, pp. 87–93, January 1994.

[14] M. Karpovsky, L. Levitin, and A. Trachtenberg, "Data veri£cation and reconciliation with generalized error-control codes," *IEEE Trans. on Info. Theory*, 2001, submitted.

[15] Y. Minsky, A. Trachtenberg, and R. Zippel, "Set reconciliation with nearly optimal communication complexity," *IEEE Trans. on Info. Theory*, 2000, submitted.

[16] 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.

[17] 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.

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

[19] 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.

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

[21] M. Karpovsky, L. Levitin, and A. Trachtenberg, "Data veri£cation and reconciliation with generalized error-control codes," *39th Annual Allerton Conference on Communication, Control, and Computing*, October 2001.

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

[23] A. Silberschatz, H.F. Korth, and S. Sudarshan, *Database System Concepts*, McGraw-Hill, third edition, 1999.

[24] "GNU Multi-precision Library," http://www.swox.com/gmp/.

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

[26] "Pilot PRC-Tools," http://sourceforge.net/projects/prc-tools/.

[27] S. Weisberg, *Applied Linear Regression*, John Wiley and Sons, Inc., 1985.

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