On the Scalability of Data Synchronization Protocols for PDAs and Mobile Devices

S. Agarwal D. Starobinski A. Trachtenberg[*]

{ska,staro,trachten}@bu.edu
Department of Electrical and Computer Engineering
Boston University

Abstract

Personal Digital Assistants (PDAs) and other mobile computing devices rely on synchronization protocols in order to maintain data consistency. These protocols operate in environments where network resources such as bandwidth, memory and processing power are limited. In this paper, we examine a number of popular and representative synchronization protocols, such as Palm's HotSync, Pumatech's Intellisync, and the industry-wide SyncML initiative. We investigate the scalability performance of these protocols as a function of data and network sizes, and compare these protocols to a novel synchronization approach, CPISync, which addresses some of their scalability concerns. The conclusions of this survey are intended to provide guidance for handling scalability issues in synchronizing data on large, heterogeneous, tetherless networks.

Keywords: Personal Digital Assistants, mobile networks, data synchronization

1 Introduction

The recent proliferation of mobile computing devices has resulted in a marked increase in the distribution of information. Under the mobile computing paradigm, users are provided access to their data wherever they are: at home on a relatively powerful and Internet-connected personal computer (PC); or on the road with a Personal Digital Assistant (PDA) or a laptop, often only weakly or intermittently connected to a network. Systems that provide such ubiquitous data access must deal with the inherent problem of *data synchronization*; that is, changes made on the road should be reflected back in the office or home and vice versa. The task of a data synchronization protocol is to quickly identify these changes, resolve possible conflicts, and propagate updates to the various synchronizing devices.

[*]Part of this work was supported by the National Science Foundation under NSF CAREER Grant CCR-0133521.

Synchronization protocols for mobile devices must be carefully designed, as the resources available for these devices are often constrained. This is especially the case with PDAs, which typically communicate over serial or wireless links with low bandwidth (10-100 Kb/s). Moreover, PDAs have limited CPU, memory and power resources and, thus, are unable to quickly process or transfer large amount of data. For example, the new top-of-the-line Palm m515 PDA is equipped with a 33-MHz Motorola Dragonball VZ processor [1] and 16 MB of RAM storage; its batteries last about one week between recharging. In a networked setting, scalable and efficient data synchronization protocols are essential for data intensive applications such as e-mail editors, spreadsheets, and even collaborative work environments.

The goal of this paper is to survey a number of representative PDA synchronization protocols and investigate their performance with respect to utilization of bandwidth, processing, and memory resources. In particular, we show that existing synchronization protocols often do not scale well with the number of devices in a network, and with the size of the data to be synchronized. We compare these protocols to a new algorithm, called CPISync [2], which addresses some of the scalability issues. Our conclusions are intended to provide guidance for addressing the scalability issues inherent to synchronizing data over large, heterogeneous, tetherless networks.

This paper is organized as follows. In Section 2, we describe some fundamentals of PDA synchronization. In Section 3, we survey, from the perspective of scalability performance, some of the most popular synchronization protocols that have been developed for mobile devices. In particular we explain Palm's Hot sync synchronization protocol [3], Pumatech's Intellisync [4] synchronization protocol, and a recent industry-wide initiative called the SyncML [5] standard. This section also includes a description of the new scalability-oriented CPISync [2] protocol. In Section 4 we provide a taxonomy of the reviewed protocols according to various scalability metrics, and consider future trends. Section 5 is devoted to concluding remarks and directions for further research.

2 Fundamentals of PDA Synchronization

A PDA can be represented by a node in an *ad hoc* network. As such, the node is intermittently connected to the network and needs to regularly synchronize with other nodes on the network to maintain the consistency of its data. This task is relatively simple when the PDA synchronizes with just one desktop machine. However, trends towards "pervasive computing" demand the ability for *multi-device synchronization*, in which a PDA synchronizes with various different desktops at home or at work, or with other mobile devices such as other PDAs, home digital assistants or cellular phones.

Single-device synchronization, where the PDA synchronizes to the same personal computer (PC) every time is simply handled by maintaining modification flags or time-stamps, as done by the "Fast Sync" mode of the Hot Sync protocol detailed in Section 3.1. Using this protocol, a PC and PDA keep track of the records that have been added, modified or deleted since their last synchronization. This information is used during the subsequent synchronization to reconcile the databases on both devices. In the case of a *conflict*, where the same record has been modified on both the PC and the PDA, the choice of which modification to keep is determined manually or chronologically.

Multi-device synchronization is more difficult, because maintaining the consistency of several

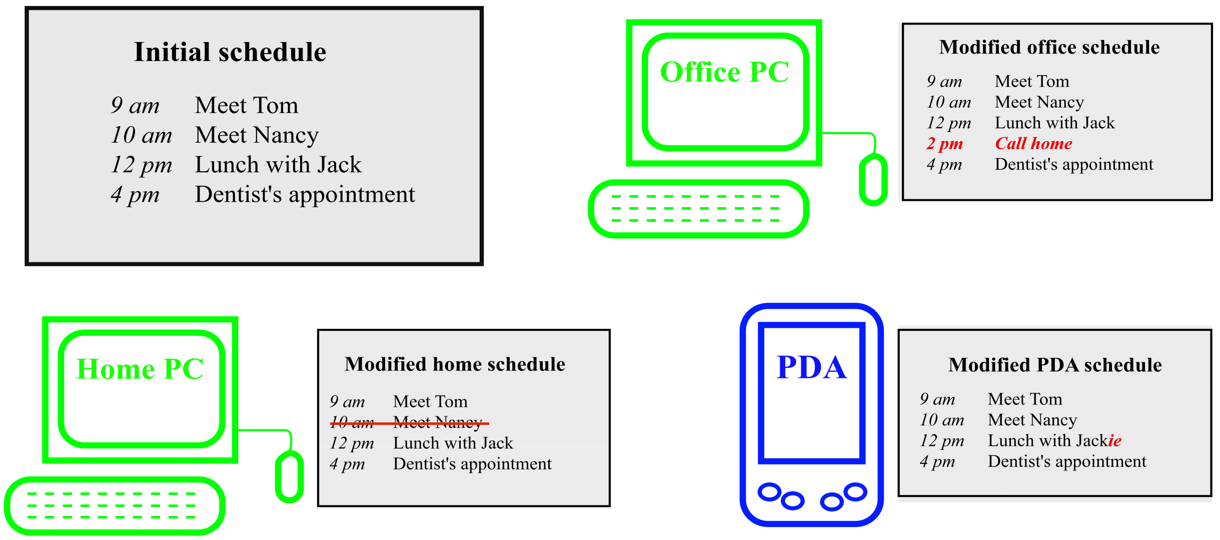

Figure 1: An appointment schedule modified on different devices: a home machine, an office machine, and a PDA.

devices might require many individual synchronizations between pairs of devices. Figure 1 shows a simple data synchronization scenario where a single initial appointment schedule is modified separately on three devices. In order for all the devices to end up with the same appointment schedule, the PDA must first synchronize with the home PC, then with the office PC, and then again with the home PC.

In a general peer-to-peer setting, the efficiency of multi-device synchronization is determined by two factors: the order in which devices reconcile their data, and the method by which these reconciliations are performed. These two factors depend in turn on the characteristics of the underlying system. For example, a synchronization schedule based on a gossip protocol [6] would have devices randomly choose partners with which to synchronize, ending (with high probability) in the complete synchronization of the data on all devices. In this scenario, the method by which two individual devices synchronize is extremely important, because its efficiency has a multiplicative effect on the overall efficiency of synchronizing the network.

3 Overview of Synchronization Protocols

We now survey several representative protocols for data synchronization, with an eye towards scalability performance. Unfortunately, details of some well-used protocols, such as Microsoft's ActiveSync, are not publicly available and could not be surveyed.

3.1 Palm HotSync Protocol

Palm PDAs, or Palm Pilots as they are popularly called, run the Palm operating system. This operating system provides an implementation of the HotSync [3] synchronization protocol as a "middleware" interface for programs whose data needs to be synchronized with other devices or

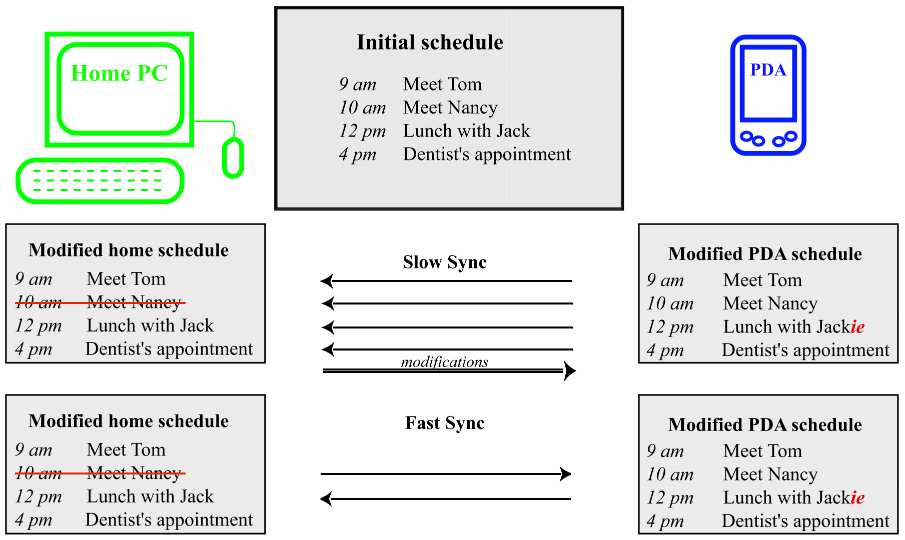

Figure 2: A comparison between the two modes of HotSync - Slow Sync and Fast Sync. Slow Sync transfers the entire database from the PDA to the PC, whereas Fast Sync transfers only modifications made after the last synchronization.

PCs. HotSync provides two modes of operations - *Slow Sync* and *Fast Sync*.

Fast Sync Fast Sync is applicable only when a handheld PDA is being synchronized with the same PC as in its previous synchronization. In this case the Palm Pilot uses status flags to determine what changes have occurred since the last synchronization. Status flags are maintained for each record in an application's database, and are toggled whenever data is inserted, deleted, or modified. Upon a synchronization request by the user, the handheld transmits to the desktop PC all records whose status flags have been changed; the PC compares these records to its own database with one of the following results:

1. The handheld record is added to the desktop database (Insertion),
2. The handheld record replaces the desktop version (Modification),
3. The record on the desktop is deleted (Deletion),
4. The record is archived on the desktop machine (Archived).

The PC then sends its corrections back to the handheld and all status flags are reset. Note that all comparisons of modified records, and subsequent decisions about what to keep, are made on the PC rather than the computationally limited handheld device. If more than two devices need to synchronize with each other, as in Figure 1, then Fast Sync cannot be used; this is because status flags would be reset when synchronizing with one device even if they are needed for synchronizing with the second device. Thus, Fast Sync trivially does not scale with the network size.

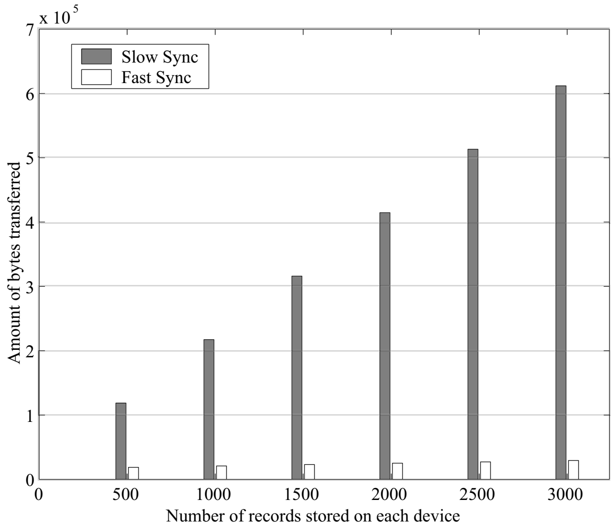

Figure 3: Comparison between the communication complexities (in bytes) of the two modes of HotSync - Slow Sync and Fast Sync. Measurements were repeated with an increasing number of 100-byte records on each device, but a fixed number of differences (*i.e.,* ten).

Slow Sync In all cases where Fast Sync conditions are not met, a Slow Sync is performed. In other words, 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. Slow Sync then refers to a backup copy of the data, which was stored on the PC during the last synchronization with the PDA, to determine which records have been modified, added or deleted. The synchronization process completes in the same way as the Fast Sync case.

Figure 2 depicts the difference between HotSync's Slow Sync and Fast Sync. In Fast Sync, the amount of communication between the PC and the PDA is reduced because only the modified records are sent from the PDA to the PC; in the Slow Sync case the entire database is transferred to the PC. For this reason Fast Sync will typically be much more efficient than Slow Sync with respect to latency and bandwidth usage.

An undesirable property of Slow Sync is that its communication and time complexities increase linearly with the number of records stored in the device, independently of the number of record modifications. Figure 3 illustrates this phenomenon on a Palm IIIxe PDA, as measured by a demonstration version of the Frontline Test Equipment software [7]. Specifically, the figure shows that the amount of data transfer needed to complete a Slow Sync grows linearly with the number of records stored in the device, whereas for Fast Sync it remains almost constant. Thus, Slow Sync does not scale with the number of entries stored on the devices. In fact, Slow Sync can require as long as 20 minutes for large, but practical, database sizes.

3.2 Intellisync

The Intellisync Anywhere [4] product family from Pumatech seeks to make every synchronization "Fast Sync enabled" in order to minimize connection times. The technology relies on a central

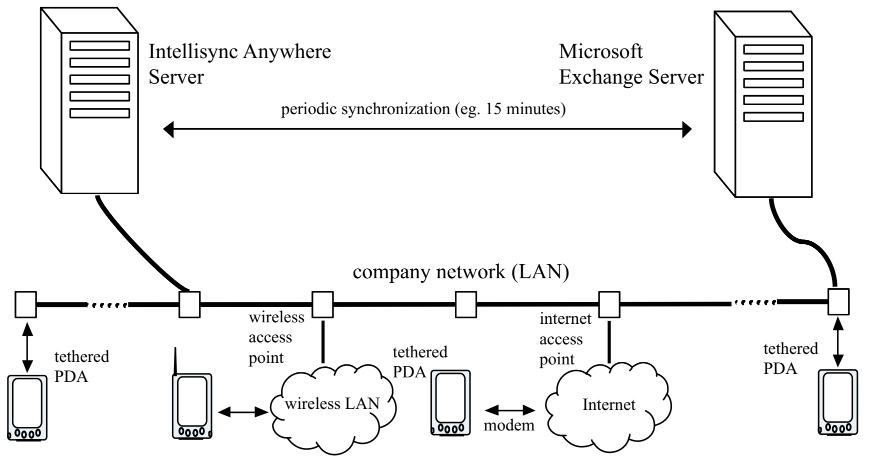

Figure 4: Intellisync Anywhere server installed on a company's network.

server, with which all devices synchronize and on which user accounts and security settings are maintained. Since each device always synchronizes with the same server, all synchronizations can be characterized as Fast Sync's. Thus the performance graphs for Intellisync will be very similar to the ones shown in Figure 3 for Fast Sync. Moreover, no peer-to-peer connections are needed to synchronize multiple devices; instead each device connects to the central server, which maintains a modification history.

The system enables synchronization of corporate applications, such as Microsoft Outlook and Exchange, between desktop and mobile devices, including Palm PDAs, Pocket PCs and Symbian handhelds. Figure 4 shows a company network using an Intellisync Anywhere server to synchronize mobile devices using the company's local area network (LAN), tethered terminals (*e.g.,* PDA cradles attached to any computer on the network), wireless LANs or the Internet.

Periodically, the Intellisync Anywhere Server synchronizes with a Microsoft Exchange server connected to fixed devices. The period between two consecutive synchronizations is controlled by the server administrator, and reflects a trade-off between the accuracy of a user's information and the communication overhead needed for frequent synchronizations.

This protocol suffers from the same problems that generally plague centralized architectures, namely a central point of failure and communication, and a lack of scalability with increasing numbers of clients. If the central server fails or becomes congested, then synchronization is affected for all clients; this problem can be mitigated with the use of several central servers, but not without introducing secondary problems in synchronizing the various servers. Moreover, no matter how the server is situated, it will necessarily be farther away from certain locations than others, translating to an increased synchronization latency.

3.3 SyncML

SyncML [5] is an open industry initiative supported by hundreds of companies including Ericsson, IBM, Lotus, Matsushita, Motorola, Nokia, Openwave and Starfish. It seeks to provide an open standard for data synchronization across different platforms and devices.

To minimize communication time, the standard assumes that each device maintains information modification flags for each of its records with respect to every other device on the network. Thus, modifying a record on a PDA would entail not just toggling one set of status flags as in Fast Sync, but rather toggling one set of status flags for every device on the network. After two devices synchronize with each other, only status flags corresponding to these two devices are cleared. In this way, each synchronization looks like a Fast Sync, with performance similar to that depicted in Figure 3.

Since a given device might synchronize with any other device, a SyncML implementation would require a device to maintain modification information about each of its records with respect to each other device. Thus, the amount of memory needed by a network of n devices each with r records would be roughly nr units per device. This memory overhead can easily become significant. Consider, for instance, a small network of $n = 20$ devices, maintaining a replicated database of $r = 10,000$ records, each associated with 8 bytes of metadata detailing modification time, status flags, etc.. The total memory overhead for this system would exceed 1 MB, which can be a substantial amount of memory for a portable device. Finally, it should be noted that timestamp protocols, such as SyncML, adapt poorly to dynamic situations where devices are frequently added or removed from the network. This is because adding or removing a device from the network entails an update to every other device in the network.

3.4 CPISync

The CPISync algorithm [8, 9], which is an abbreviation for Characteristic Polynomial Interpolation Synchronization, is based on an algebraic solution to the problem of reconciling two remote sets of information. This algorithm has been shown [2] to be significantly more efficient than Slow Sync in the most common scenario of data synchronization - where the number of differences to be reconciled is far smaller than the sizes of the reconciling databases. This scenario is typically the case for PDA applications, where changes to an address book, for example, typically consist of only one or two entries, whereas the address book might contain hundreds of entries in total.

Unlike the previous protocols, the CPISync algorithm is computationally intensive (time nearly cubic in the number of differences between synchronizing devices), though almost all the computation can be done on one device. Thus, in a PDA - PC synchronization, the computational component of the protocol can be shifted to the relatively fast PC.

One feature of the CPISync algorithm is that it maintains the communication efficiency of Fast Sync in a general, peer-to-peer setting. The amount of data that it transfers for a synchronization depends linearly on the number of differences between databases, and is for all intents and purposes independent of the overall sizes of the databases. Unlike the previous protocols, CPISync does not need to maintain any state information about any other devices on a network, and it can function in a peer-to-peer fashion in an ad hoc environment. In effect, CPISync can truly operate as a middleware interface to any protocol that maintains consistency of distributed information.

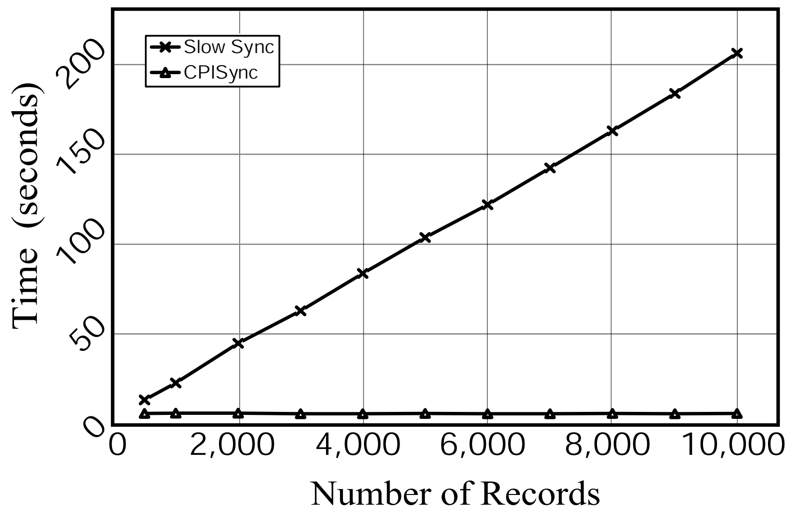

Figure 5: Time needed to synchronize with CPISync versus Slow Sync.

Protocol Overview The CPISync algorithm represents databases of records by sets of integers, with each integer representing a record. A set $\{x_1, x_2, x_3, \ldots, x_n\}$ is represented by a *characteristic polynomial*

$$\chi_S(Z) = (Z - x_1)(Z - x_2)(Z - x_3)\ldots(Z - x_n).$$

The key to CPISync is the observation that for two sets S_A and S_B,

$$\frac{\chi_{S_A}(Z)}{\chi_{S_B}(Z)} = \frac{\chi_{\Delta_A}(Z)}{\chi_{\Delta_B}(Z)}, \tag{1}$$

where Δ_A represents the set difference $S_A - S_B$, and similarly $\Delta_B = S_B - S_A$. This is because terms common to both sets cancel out in the numerator and denominator. Thus, the rational function in 1 can be uniquely interpolated from $\overline{m}$ samples, if there are at most $\overline{m}$ differences between synchronizing sets.

The CPISync algorithm can be generally described as follows:

1. Hosts A and B evaluate their characteristic polynomials on $\overline{m}$ sample points (over a chosen finite field). Host A sends its evaluations to host B.
2. The evaluations are combined to compute $\overline{m}$ sample points of the rational function $\frac{\chi_{S_A}(Z)}{\chi_{S_B}(Z)}$, which are interpolated to determine $\frac{\chi_{\Delta_A}(Z)}{\chi_{\Delta_B}(Z)}$.
3. The numerator and denominator of the interpolated function are factored to determine the differences between S_A and S_B.

Experimental data Figure 5 shows the superior scalability (in terms of synchronization time) of CPISync as compared to Slow Sync, for a fixed number of differences and two devices. The results were obtained for a Palm IIIxe PDA synchronizing with a Pentium III class machine. The

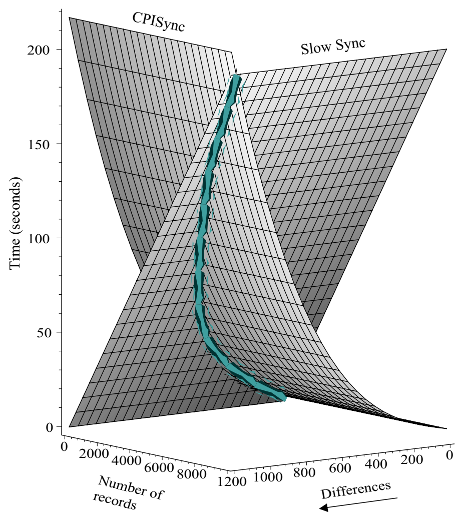

Figure 6: Synchronization time of Slow Sync versus CPISync. The intersection of the two surfaces marks the threshold values at which Slow Sync and CPISync require equal amounts of time.

CPISync time does not grow with the number of records on the device, whereas Slow Sync time grows linearly. For large databases and large numbers of synchronizing devices, this difference in time and communication is substantial, rendering Slow Sync effectively unusable for such environments.

The nearly cubic computation complexity of CPISync implies that there is a threshold number of differences below which CPISync is faster than Slow Sync and above which Slow Sync is faster than CPISync. This threshold can be determined analytically for a particular system [2], and then used in a hybrid scheme for efficient synchronization in a general setting. Note that the value of the threshold is typically quite large, making CPISync the protocol of choice for many synchronization applications. For instance, for $1,000$ records, this threshold corresponds to about 400 differences, meaning that if up to 400 records differ between synchronizing devices, then CPISync will be the faster algorithm.

Figure 6 depicts an analytical computation of this threshold. It shows that Slow Sync time increases linearly with the size of the database whereas CPISync time remains essentially constant as the database size increases. On the other hand, CPISync time increases rapidly with increasing number of differences between the two synchronizing databases whereas Slow Sync remains fairly constant. Recent results [10] modify CPISync to avoid cubic computational growth at the expense of multiple rounds of communication.

4 Taxonomy of Synchronization Protocols

In this section, we first provide a qualitative comparison of the scalability performance of the various protocols discussed in this work. We then compare the suitability of these protocols to different applications and consider future trends.

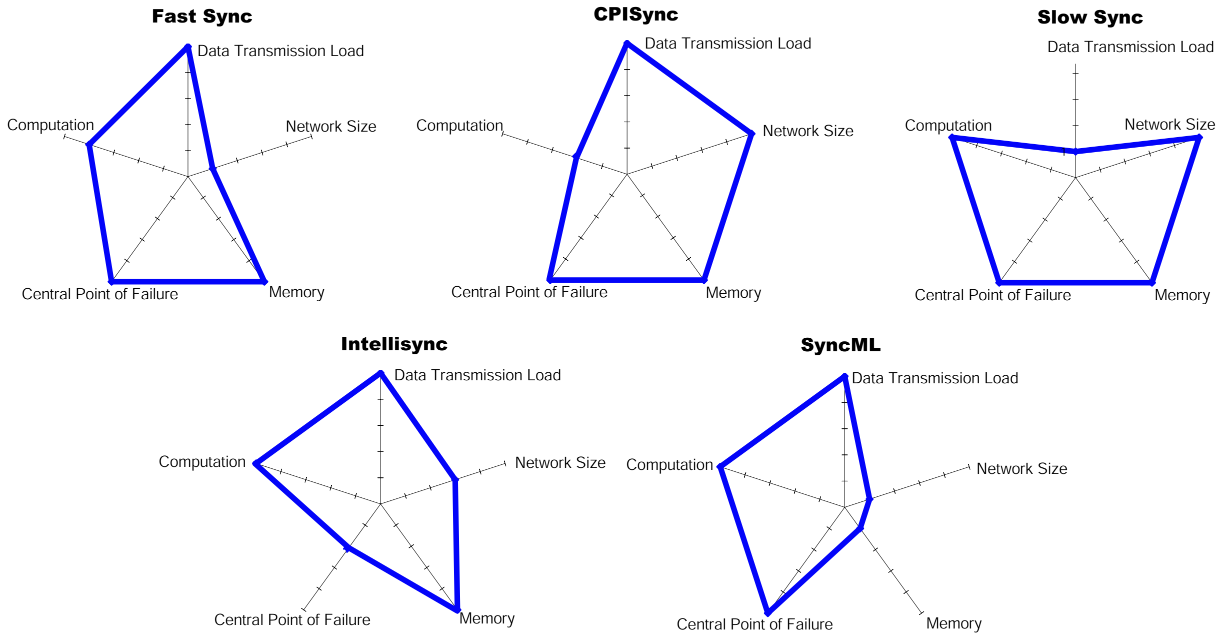

Figure 7: Scalability strengths and weaknesses of the different synchronization protocols. An absolute scale is used in which high values denote strengths and low values denote weakness.

4.1 Scalability

Each of the synchronization protocols described in the previous section addresses different scalability concerns. The "radar" diagrams in Figure 7 qualitatively compare these protocols, highlighting their strengths and weaknesses. Each protocol is judged according to the following criteria:

Data Transmission Load This refers to the amount of data that is exchanged between synchronizing devices. It is a particularly important metric of scalability because it directly affects the time needed for synchronization. CPISync requires only a little more communication than the information theoretic minimum needed for synchronization [9].

Computation This refers to the amount of processing done by devices when they synchronize. Computational capabilities of PDAs and mobile devices are generally relatively weak, so this is an important metric. Slow Sync is one winner on this criterion because it requires almost no computation. CPISync, on the other hand, is computational intensive, but most of its computation is asymmetric and can be done on the more powerful synchronizing device.

Network Size This refers to the ability of the protocol to scale with increasing numbers of nodes in the network. Fast Sync is the weakest in this category because it only allows two hosts to synchronize. SyncML is also unsuitable for large networks because each device has to maintain synchronization information with respect to every other device in the network. Intellisync, though better, is still not very scalable with network size because its central server takes all the synchronization load.

PDA *Storage Capacity*	**Palm Pilot** *1MB*	**Palm m505** *8MB*	**iPAQ, HP Jornada** *64MB*	**Apple Ipod** *5GB*
RS-232 @ 115kb/s	1 minute	10 minutes	1 hour	4 days
USB 1.0 @ 12Mb/s	1 second	6 seconds	1 minute	1 hour
Ethernet @ 100Mb/s	80 ms	1 second	5 seconds	7 minutes
Firewire 1394 @ 400Mb/s	20 ms	160 ms	1 second	2 minutes
USB 2.0 @ 480 Mb/s	18 ms	150 ms	1 second	1 minute
Firewire 1394b @ 800 Mb/s	10 ms	80 ms	1 second	1 minute

Table 1: Approximate time required to perform wholesale data transfer (Slow Sync). Recent data transfer technologies such as USB and Firewire have reduced data transfer times by orders of magnitudes, though the potential size of data that may be transferred to or from a PDA has also increased accordingly.

Robustness This is a desirable feature of any multi-device synchronization protocol. Intellisync is not very robust because its server presents a central point of failure. The other protocols are more distributed and so the risk of synchronization failure is significantly lower.

Memory This relates to the amount of memory overhead needed by each synchronization protocol. SyncML is the least scalable protocol from this perspective. Slow Sync is scalable as long as PDAs synchronize only with PCs, whose large memory can usually accommodate a local comparison of two synchronizing databases.

Each of the synchronization methods excels in scalability within a particular framework and for specific categories. Fast Sync works best for two-device systems, whereas Slow Sync is suited for computationally weak networks with very fast interconnections. Intellisync and SyncML are suitable for small to mid-sized networks, and CPISync is best suited for large networks with relatively slow interconnections. The ideal protocol, corresponding to a full pentagon in the radar graphs of Figure 7, can probably be forged out of a hybridization of these various protocols.

4.2 Scenarios and Trends

The HotSync protocol lends itself well to scenarios where synchronization always takes place between the same two devices, thus allowing for activation of the Fast Sync mode. Otherwise, it may be useful only when the size of the synchronizing databases is small (relative to the link speed) so that the data transfer time of Slow Sync is not a major constraint. Typical current usage of PDAs, in which synchronization only occurs with one PC, is thus well matched to HotSync.

It is interesting to note that the latency problem of Slow Sync may not necessarily be solved by the use of faster connection interfaces, such as Firewire (IEEE 1394) [11] and USB [12], due to the corresponding increase in the storage capacity of handheld devices. Table 1 compares the time required to perform a wholesale data transfer for various representative PDAs and interface technologies. We observe that a Palm Pilot (1MB storage) using a serial link to transfer its entire data to a PC will require approximately the same order of time (*i.e.,* one minute), as a state-of-the-art Apple Ipod (5GB storage) using the Firewire 1394b data transfer technology.

We expect centralized architectures, such as Intellisync, to provide good solutions for small to med-sized corporate networks. For example, solutions based on the Intellisync technology

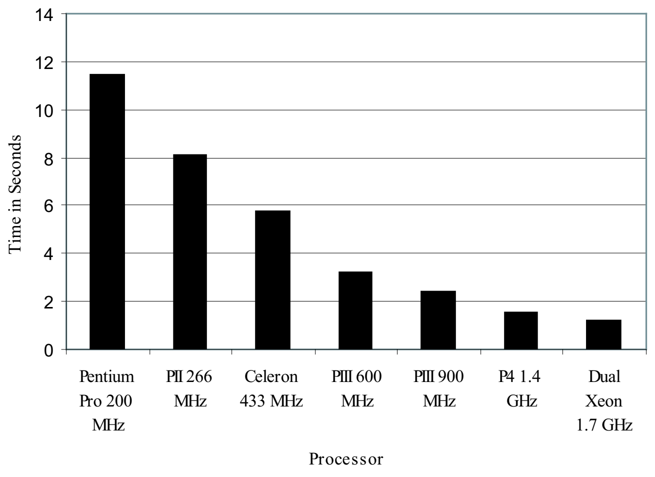

Figure 8: Computation time of CPISync using various processors for the reconciliation of 200 differences between two databases of arbitrary size.

have been deployed to allow school administrators of the New York City Board of Education to efficiently synchronize e-mail and calendar information between their Palm OS handhelds and the New York Board of Education's Microsoft Exchange server.

The SyncML standard will allow integration of a heterogeneous mix of devices manufactured by different companies. For instance, IBM recently unveiled the SyncML-based DB2 Everyplace relational database to manage data in PDAs and embedded devices. The synchronization architecture of DB2 Everyplace [13] is based on a central server, making it akin to the Intellisync architecture. We believe that SyncML will also be useful in peer-to-peer settings. However, its memory overhead may become a serious limitation in large networks.

The CPISync protocol is a future technology. The primary strength of this protocol is its ability to perform synchronization with an almost minimal amount of communication. CPISync is, thus, a strong candidate for wireless synchronization and dynamic peer-to-peer settings. With increasing CPU capabilities we believe that computation will cease to be a serious bottleneck for CPISync, as shown by Figure 8. Though we have not implemented CPISync between two PDAs as yet, newer PDAs, such as those based on StrongARM [14] processors, will most probably run CPISync with acceptable computation times. For example, some commercial models of the iPAQ Pocket PC based on StrongARM architectures run at clock speeds of 206 MHz, approaching the lower end of the clock speeds shown in Figure 8. For applications with low round-trip communication latency, the latest results in [10] would eliminate the computational bottleneck of CPISync altogether.

5 Conclusion

In this paper, we have surveyed a number of synchronization protocols available for synchronizing PDAs with other mobile devices and with wired PCs. Each of the protocols has its strengths and weaknesses, arbitrating scalability in three parameters: (i) the amount of information exchanged between two devices during synchronization, (ii) the amount of information that needs to be stored in the memory of the devices to allow consistent synchronization, and (iii) the computational complexity of the synchronization. In some applications with large propagation delays between hosts, the number of rounds of communications per synchronization, which is to say that the number of times there is a two-way exchange of communication between devices, may also be an important factor.

The optimum choice of synchronization protocol depends largely on the application and the type of data being synchronized. For example, if data changes slowly relative to the frequency of synchronizations, then CPISync is the most scalable solution. On the other hand, if data changes frequently and network connections are very fast, then Slow Sync should be used. Therefore, an important area of further research is the development of statistical models for user activities that predict how often users synchronize their data and what amount of data is typically updated during synchronization events. The development of such models would require collection and processing of real data from users.

This paper has shown that the development of scalable data synchronization protocols is of critical importance for the deployment of large-scale mobile computing platforms. We expect that the conclusions of this paper will highlight scalability issues for future-generation tetherless systems.

References

[1] Motorola. Dragonball : Enabling the next generation of wireless devices. http://e-www.motorola.com/collateral/DBBROCHURE.pdf.

[2] A. Trachtenberg, D. Starobinski, and S. Agarwal. Fast PDA synchronization using characteristic polynomial interpolation. In *IEEE Infocom*, 2002. To appear.

[3] Palm developer knowledgebase manuals. http://palmos.com/dev/support/docs/palmos/ReferenceTOC.html.

[4] Pumatech Whitepapers. Invasion of the data snatchers, 1999. http://www.pumatech.com/enterprise/wp-1.html.

[5] SyncML specifications. http://www.syncml.org/downloads.html.

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

[7] Frontline test equipment. http://www.fte.com.

[8] Y. Minsky, A. Trachtenberg, and R. Zippel. Set reconciliation with nearly optimal communication complexity. Technical Report TR1999-1778, TR2000-1796,TR2000-1813, Cornell University, 2000.

[9] Y. Minsky, A. Trachtenberg, and R. Zippel. Set reconciliation with nearly optimal communication complexity. In *International Symposium on Information Theory*, page 232, June 2001.

[10] Y. Minsky and A. Trachtenberg. Practical set reconciliation. Boston University Technical Report 2002-03.

[11] IEEE 1394. http://computer.org/multimedia/articles/firewire.htm.

[12] Universal serial bus. http://www.usb.org/developers/docs.html.

[13] Angela Noether. Extending enterprise data to mobile devices. http://www-3.ibm.com/software/data/db2/everyplace/extendingenterprise.pdf.

[14] Intel PCA application processors. http://developer.intel.com/design/pca/applicationsprocessors/index.htm.