BOSTON UNIVERSITY

COLLEGE OF ENGINEERING

Thesis

DATA SYNCHRONIZATION IN MOBILE AND DISTRIBUTED NETWORKS

by

SACHIN KUMAR AGARWAL

B.Tech., Regional Engineering College, Warangal, India
May 2000

Submitted in partial fulfillment of the
requirements for the degree of
Master of Science
2002

Approved by

First Reader

Ari Trachtenberg
Assistant Professor of Electrical and Computer Engineering
Boston University

Second Reader

David Starobinski
Assistant Professor of Electrical and Computer Engineering
Boston University

Acknowledgments

I would like to first acknowledge my advisor, Prof. Ari Trachtenberg for his constant support and guidance through the research and writing that constitute this thesis. It is a great understatement to say that this work would not have been possible without his help. I would also like to extend my gratitude to Prof. David Starobinski for his constructive inputs in the form of discussions and new ideas to this work. I would also like to acknowledge my readers, Prof. David Starobinski and Prof. Jeffrey Carruthers for their comments and helpful discussions.

I would like to thank my family who has been extremely supportive throughout the time I've been working on my thesis. My father and mother for being there when I needed them and my brother for his encouragement.

Finally I would like to thank my lab cohorts for their help and tolerance during the thesis process.

DATA SYNCHRONIZATION IN MOBILE AND DISTRIBUTED NETWORKS

SACHIN KUMAR AGARWAL

Boston University College of Engineering, 2002

Major Professor: Ari Trachtenberg, Assistant Professor of Electrical and Computer Engineering

Abstract

The rapid increase in networked mobile devices has made it important to develop scalable data synchronization protocols that will periodically synchronize data held on these devices. Synchronization seeks to maintain consistency in data that is being changed on each of these mobile hosts independently. This has to be achieved within practical constraints on overhead that limit the amount of data exchanged during synchronization, the amount of memory used to store synchronization data, and the computation involved while running the synchronization protocol. In addition, any synchronization protocol should scale with the number of devices that might be synchronized with each other and should be resilient to failure, given the ad-hoc nature of mobile networks.

We study some of the representative synchronization protocols in use today and then compare them to characteristic polynomial interpolation synchronization (CPISync), a more mathematical approach to synchronization implemented by us on a PC-PDA synchronization system. Two devices synchronizing using CPISync exchange only $O(\overline{m})$ bits where $\overline{m}$ is the upper bound on the number of differences between the reconciling data sets of the devices. Thus, CPISync is well-suited for

typical PC-PDA data synchronization scenarios where the number of changes (additions/deletions/modifications) to databases between two synchronizations is typically bounded by a small constant. When a good bound on $\overline{m}$ is not known a priori we show that an enhancement to the basic algorithm keeps CPISync time complexity within a small multiplicative factor of the case when $\overline{m}$ is known a priori. Our experiments show that CPISync scales well with growing data size, which is increasingly the case with newer mobile devices.

Contents

Abstract **iv**

Table of Contents **vi**

List of Tables **viii**

List of Figures **ix**

1 Introduction **1**

1.1 The Broad Picture 1

1.2 Applications of Synchronization 2

1.3 Scope of this Thesis 4

2 Contemporary Synchronization Technologies **8**

2.1 Conflicts 9

2.2 Scalability in Mobile Device Networks 10

2.2.1 Hotsync 11

2.2.2 Intellisync 14

2.2.3 SyncML 16

2.2.4 CPISync 17

2.3 Putting it together 19

3 PDA Synchronization **22**

3.1 CPISync 23

3.2 Deterministic CPISync 23

3.2.1 Metrics and Measurements 32

3.2.2 Results 33

3.3 Probabilistic CPISync 38

3.3.1 System Model 40

3.3.2 Results 43

4 Conclusions **45**

4.1 Summary 45

4.2 Future Work 47

Curriculum Vitae **56**

List of Tables

3.1 Threshold values at which CPISync takes the same time as Slowsync. 36

3.2 Fitted polynomials $P(m)$ for CPISync running time for various data set sizes. 37

List of Figures

1.1 Representative PDA Specification - Palm Vx 3

1.2 Some applications of synchronization 5

2.1 Synchronization and Conflicts 9

2.2 Pentagon of Scalability in Mobile Device Networks 10

2.3 Communication complexities compared - Slowsync and Fastsync . . . 12

2.4 Slowsync and Fastsync modes of Hotsync 14

2.5 Newer data transfer technologies and memory sizes 15

2.6 Intellisync Anywhere server installed on a company's network 16

2.7 Scalability of Slowsync and CPISync with database size and number of differences 18

2.8 Synchronization protocols - a comparison 21

3.1 CPISync explained by example 25

3.2 Finite fields in CPISync 27

3.3 CPISync PC-PDA synchronization model 29

3.4 Sample experiment script 31

3.5 Scalability of CPISync and Slowsync 32

3.6 Experimental results - CPISync vs. Slowsync for small data sets . . . 34

3.7 Experimental results - CPISync vs. Slowsync for large data sets . . . 35

3.8 Flowchart showing steps in Probabilistic scheme 38
3.9 Latency optimization for Probabilistic scheme 41
3.10 Probabilistic scheme vs. Deterministic scheme 43

4.1 Name-dropper communication complexity using CPISync and whole data transfer 48
4.2 Name-dropper time complexity using CPISync and whole data transfer 49

Chapter 1

Introduction

1.1 The Broad Picture

Dynamic information is the currency of most computer networks today. Information available on networked devices tends to change over a period of time, rather than remain static, and these changes must be propagated throughout the network. While the specific requirements of propagating these changes may vary from real time dissemination of changes to lazy propagation of changes, there have to be mechanisms which guarantee this functionality, at an *acceptable overhead*.

An ad hoc network is an network of mobile routers and hosts that together form an arbitrary network topology. The routers and hosts in an ad hoc network are free to move and organize themselves arbitrarily; thus, the network's topology may change randomly. Mobile hosts may be considered as weakly connected nodes in an ad hoc network [1] because of the absence of a fixed topology connecting these devices and the intermittent addition and removal of such devices from the network. Note the ambiguous use of the term "network" - the other end of the network to which the device is synchronizing may be a mobile device, a PC or a server. For example, an

appointment schedule may be simultaneously maintained on a PC in office, a PC at home, and a mobile device like a PDA (Personal Digital Assistant). The three hosts will have to synchronize the appointment database periodically to maintain consistent data in the database.

A natural metric of *acceptable overhead* for synchronization will usually be a measure of the cost in terms of the latency and the bandwidth involved in dissemination of information through the network for synchronization. Mobile device network throughput is also very sensitive to the number of rounds involved in any communication. Non-interactive synchronization protocols that are not dependent on repeated data exchanges while synchronizing are preferred because the overhead of packet transmission is substantial for wireless networks (increasingly used by mobile devices) as compared to wired networks. Mobile devices are usually also bogged down due to excessive protocol overhead because of their limited computational capabilities and limited power resources. Some of these limitations become apparent from Figure 1.1, which highlights the specifications of the Palm Vx PDA [2]. For example, the Palm Vx has a single threaded processor with a clock speed of just 32MHz, which is lesser than even entry level PCs by orders of magnitude.

1.2 Applications of Synchronization

The same data may be separately modified on hosts and these changes need to be propagated to other hosts on the network . A good example is the Internet Domain Naming System (DNS) described in [3, 4]. DNS is used to map Internet address strings into IP addresses. It has an elaborate hierarchial system of updating the domain name databases periodically (typically 3 hours) because DNS lookup tables change with time. The Internet is divided into various DNS zones. Each zone has

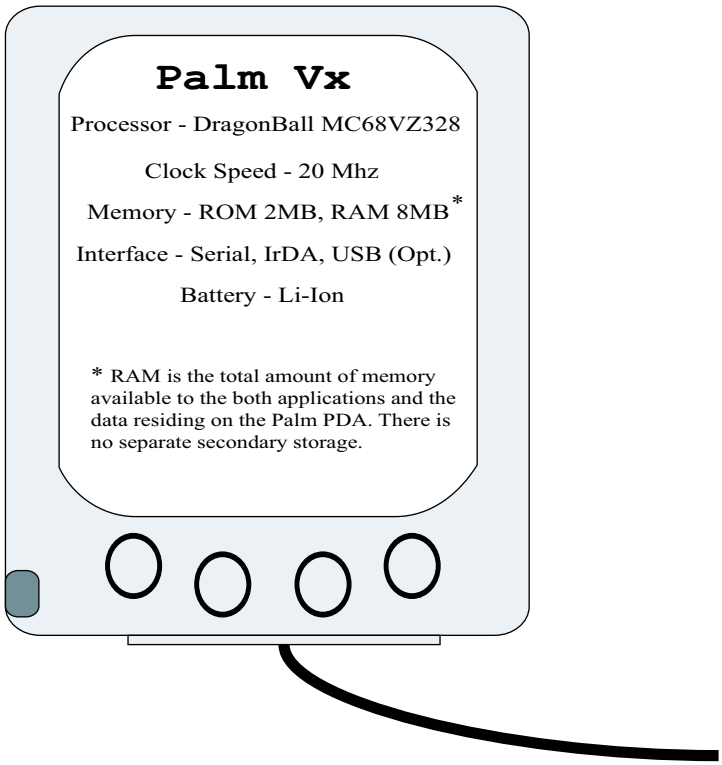


Figure 1.1: Representative PDA Specification - Palm Vx

a primary DNS server and one or more secondary DNS servers that periodically synchronize DNS information with the primary DNS server in order to obtain the latest version using either all zone transfer (AXFR) [3, 4] or incremental zone transfer (IXFR) [5] zone transfer. AXFR zone transfer results in the primary DNS server sending its whole DNS database to the secondary server and uses a lot of network bandwidth. IXFR zone transfer on the other hand is incremental and only changes in the DNS database are sent from the primary server to the secondary server. Each DNS database version has a version number that is used by the primary server to determine the updates to send to a secondary DNS server. This requires the primary server to store many previous versions of the DNS database to determine which updates are to be sent to the secondary server with a given version number. Information about older versions should be purged if the total length of an IXFR

response would be longer than that of an AXFR zone transfer, in which case it would be more efficient to do a complete AXFR zone transfer. Quick and efficient updates to the DNS information can substantially reduce the problem of Internet "Hot Spots" [6] which arise when there is a sudden increase in the number of hits on a particular web site. Hotspots overwhelm a particular server because of a large number of HTTP requests. This situation can be corrected by quickly changing the DNS information held on name servers so that alternate servers help balance the load.

There are other applications that require synchronization. Unix utilities like rsync [7] are used to synchronize files with minimum transfer of redundant information. The Coda System [8, 9] proposed by Kistler and Satyanaraynan *et al* provides a model for disconnected operation and subsequent reconciliation of distributed file systems and data in mobile devices. The general Coda system model is similar to the one used in some mobile device networks, including some PDAs. An efficient synchronization routine will be useful in boosting the performance of resource discovery algorithms such as the Name Dropper algorithm described in [10]. In addition, there is a pressing need for efficient synchronization in gossip information dissemination such as in [11]. There are countless other applications of synchronization, some of the more important being routing information dissemination in networks, synchronization of mobile device databases and the distribution of software updates over the Internet. Figure 1.2 shows some applications where data synchronization is critical.

1.3 Scope of this Thesis

There are two basic components to this thesis. Chapter 2 of this thesis, which is based on [12], discusses some of the contemporary synchronization technologies

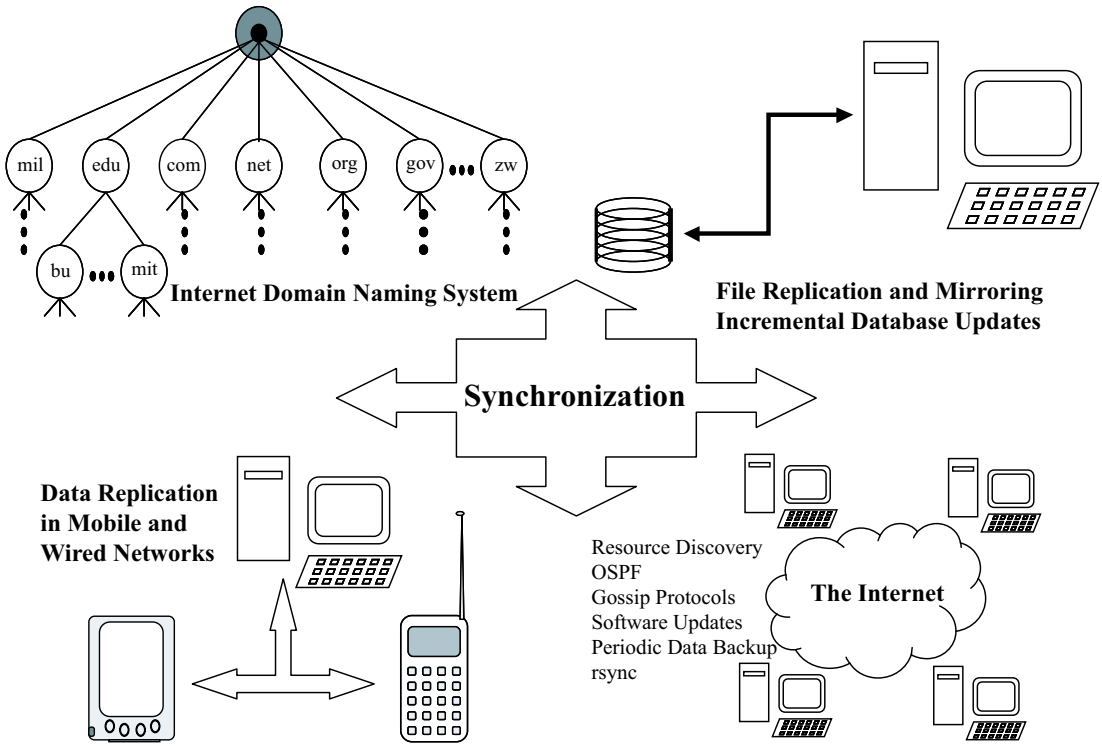


Figure 1.2: Some applications of synchronization

and compares their performance with respect to utilization of bandwidth, processing and memory. We have discussed Hotsync (the synchronization scheme used in Palm PDAs), Pumatech's Intellisync and the industry wide SyncML initiative. In addition, there is a brief description of the CPISync algorithm in Chapter 2 to compare it with the above protocols. The survey brings out some of the scalability limitations of each these protocols.

In the second part, we discuss CPISync, which is a synchronization solution based on a set reconciliation problem's solution. Chapter 3 describes the CPISync algorithm in detail and application of CPISync algorithm for PDA synchronization. Some of this work has appeared in [13]. This algorithm was first proposed in [14] in which the authors had provided preliminary results that suggested that the algorithm could be useful for data synchronization among databases with a limited number of differences

between them.

The major contribution of our research is transcribing the algorithm from a purely mathematical idea to an actual implementation on a mobile device (Palm PDA) and a PC. The algorithm was studied and analyzed first. A substantial part of the research was the partitioning of the algorithm into two asymmetric parts (in terms of computational complexity) corresponding to the asymmetric computational capabilities of a PDA and a PC. The less computationally intensive part was coded into the *slower* Palm IIIxe PDA running the Palm Operating System while the more intensive part of the algorithm was coded into the *faster* PC. The design specification took into consideration subtleties like fitting all the calculations to the word size of the Palm processor architecture to speed up the computation on the Palm PDA. The Palm PDA has a word length of 16 bits and methods to limit all mathematical operations within this limit were put in place. We also implemented a more advanced 512 bit data version of CPISync in anticipation of future applications of the algorithm not necessarily related to PDAs. In all these implementations, we tried to reduce the constants associated with the cubic complexities of some of the mathematical operations of CPISync. In addition to all this, issues of communication between the Palm PDA and the PC were sorted out and put in place for a real implementation on the PC-PDA synchronization system.

Methods and tools to quantify and compare the performance of CPISync with the Hotsync protocol (the commercially used synchronization protocol in Palm PDAs) were developed and deployed. Results from this analysis were used to mathematically model an intelligent mechanism to switch between CPISync and Hotsync on the specific requirements of a synchronization. These results were also used to design a more sophisticated version of CPISync which could synchronize without a prior bound on the number of differences between the databases. This was the 'probabilistic' algo-

rithm first proposed in [14]. The more sophisticated version was also implemented on the PC-PDA system. In related work, our implementation of CPISync is being used as the underlying synchronization routine in the development of a real memo application on the Palm PDA platform. The memo application uses CPISync to synchronize with PCs, as described in [15].

The encouraging results of the utility of CPISync in fast PDA synchronization lead us to apply the algorithm to the more general problem of Network Synchronization. We concentrated on the resource discovery problem in networks in particular. To this end, we picked upon the 'Name-Dropper' network resource discovery algorithm [10]. We built a simple simulator to compare the performance of the Name-Dropper using CPISync and wholesale data transfer (like Slowsync) separately for synchronization. For completeness, we contrasted these results with flooding, a commonly used resource discovery algorithm. This project brings forward some interesting results and we hope that more work in this area will be an interesting extension to the application of CPISync.

Finally there is a conclusion at the end of this thesis, which includes directions for future work, including the preliminary work on the application of CPISync to the network resource discovery problem.

Chapter 2

Contemporary Synchronization Technologies

Data synchronization is the process of reconciling two sets of data (databases) so that their contents are identical at the end of synchronization. The end result of a device synchronizing with another device would be the emergence of an identical copy of the database on both the devices that reflects the data that was common to both the devices as well as unique data that each of the device contributed.

In this chapter we first describe the concept of a 'conflict' in synchronization in Section 2.1. In Section 2.2 we explain synchronization issues with mobile devices in particular and then go on to explain some synchronization technologies like Hotsync, Intellisync, SyncML and finally CPISync. The description of CPISync is brief here and is meant to serve as background for Section 2.3 that compares the various synchronization protocols mentioned above. Some of the work presented in this chapter appeared in [12].

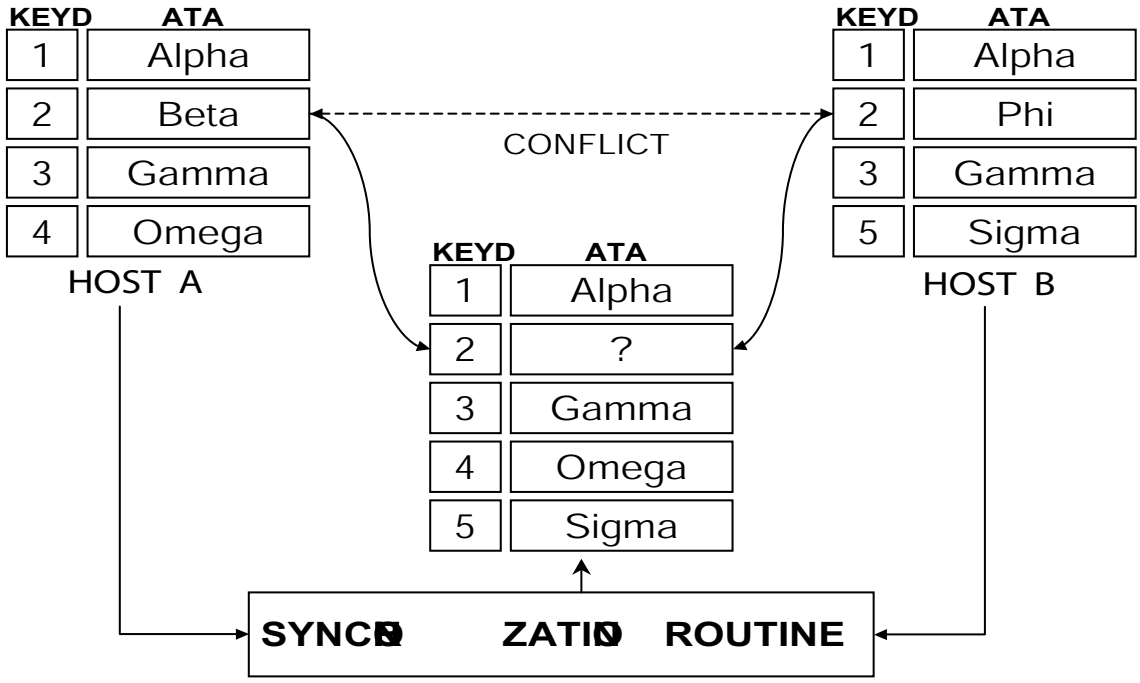


Figure 2.1: Synchronization and Conflicts

2.1 Conflicts

Inherent to any data synchronization routine is the possibility of a *conflict*. Conflicts arise when the same record (a record identified by the same key) is modified locally on different hosts. In such cases the synchronization routine has to make a decision about which data to pass into the new synchronized database and which data to ignore. Often, decisions are made on criteria like keeping the more current data (using time stamps) or by a user specified preference. For example, a particular user who makes changes to his appointment schedule on his PDA while his secretary makes changes to his appointment schedule on an office PC might want that the PDA data (updated by him) always overrides the PC data (updated by his secretary) in case there is a conflict.

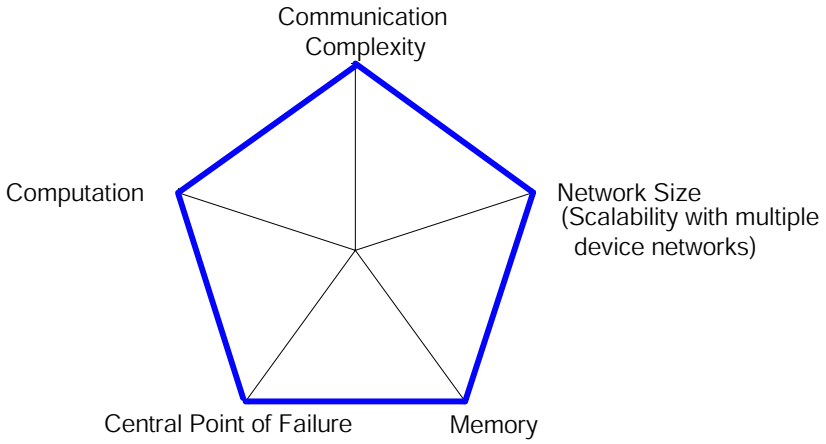


Figure 2.2: Pentagon of Scalability in Mobile Device Networks

An instance of conflict is shown in Figure 2.1. The record with key value 2 in the database held by Host A and Host B has different data. It is up to the synchronization routine to decide what value the data field pointed by the key 2 should be written to synchronized database, as shown in the figure. These decisions are resolved manually in case there is no rule set by the user about which data should be overwritten.

We will not address conflict resolution issues here, a good discussion of which can be found in [16].

2.2 Scalability in Mobile Device Networks

The overall end user synchronization experience in a mobile device is determined by a variety of factors including latency in synchronization, the amount of battery power used, any monetary cost involved (such as the cost of the network connection, air time etc.) and the robustness and reliability of the synchronization routine. The overall scalability of any synchronization protocol is determined by how well the protocol addresses these issues.

PDAs and other mobile devices are limited in how fast they can communicate over a network because of the nature of the links that connect them. Moreover, their limited computational capabilities are a significant factor in the design of a synchronization protocol for these devices. They have additional constraints of limited memories and limited power supplies (see Figure 1.1). Figure 2.2 shows the overall issues when mobile device synchronization is considered, with each of the vertices of the pentagon representing a scalability issue in mobile device synchronization. The computation and communication have a direct bearing on the latency and the battery usage of the device. Mobile device memories are limited and any protocol which uses a lot of memory will not be suitable for many of the lower end devices. At the same time, we wish to develop protocols that allow a large number of devices on the network to synchronize among themselves and are robust and preferably peer-to-peer instead of centralized, keeping in line with the ad-hoc nature of mobile device networks.

2.2.1 Hotsync

Palm PDAs run the Palm Operating System that provides the Hotsync protocol to synchronize databases held on the PDA and the PC to which the PDA synchronizes. Hotsync operates in two modes, depending on the synchronization history of the PDA with the particular PC.

A **Fastsync** occurs when the PDA synchronizes with the same PC its synchronized with the last time. In this case the PDA synchronizes by processing records based on the status flags of each record which indicate whether it has been modified, deleted, left untouched or is a new record. This method approaches the lower information theoretic bounds on the communication complexity because intuitively

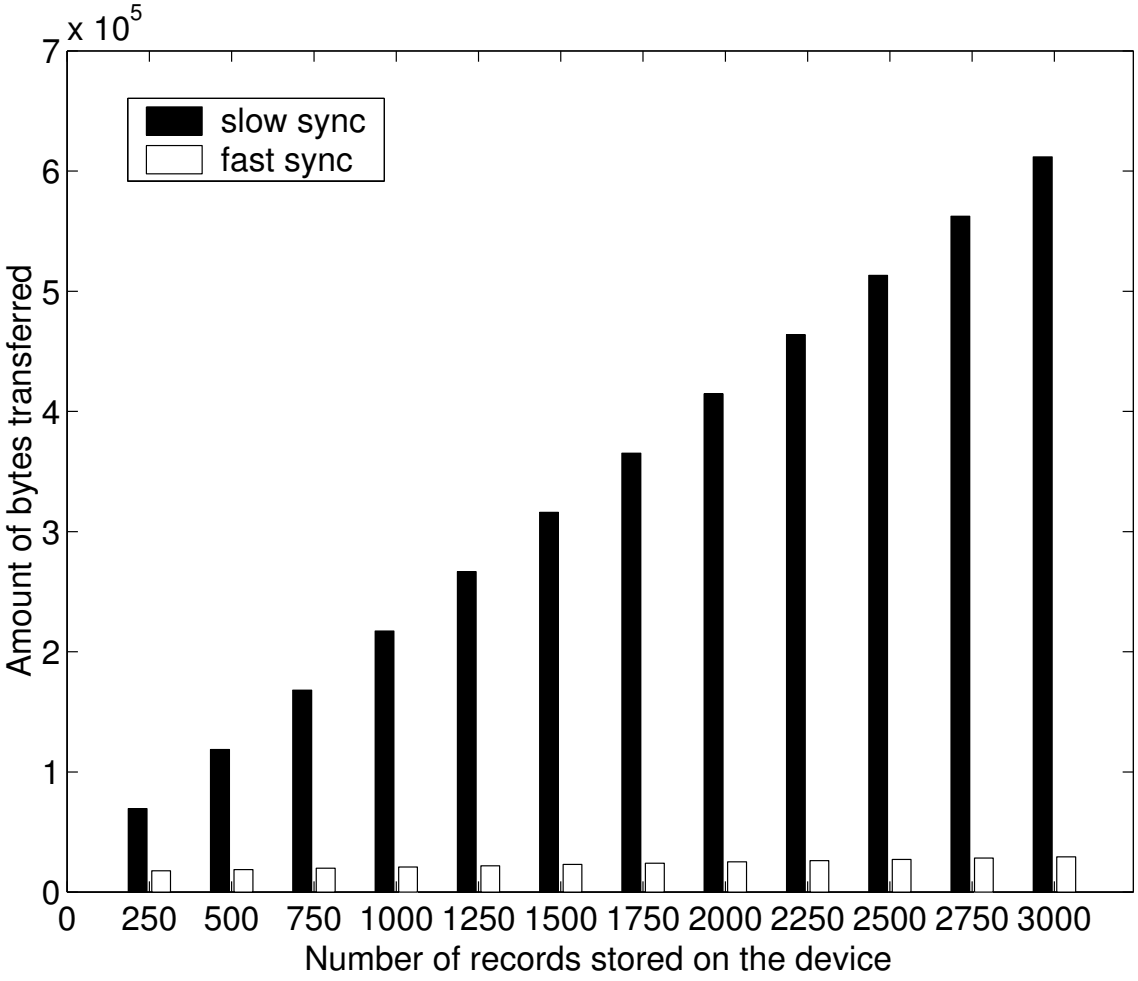


Figure 2.3: Comparison between the communication complexities (in bytes) of Fast-sync and Slowsync.

the amount of data to be sent across to synchronize is at least equal to the difference between the two data sets (provided we knew somehow which data was updated or was new in the hosts with respect to the other). A more rigorous information theoretic view of this concept is explained in [14] which suggests that the lower bound on the amount of information to be exchanged between hosts A and host B with m differences between their reconciling data sets comprised of b bit elements is given by

$$I_{trans} \gtrapprox b \cdot m - m \cdot log(m) \tag{2.1}$$

This problem of reconciling two hosts' data sets is formally known as the set reconciliation problem.

Thus Fastsync is the 'best option' for two devices that synchronize with only each other. However, Fastsync will not scale with the number of devices in the network because the PDA can remember status flags only with respect to one PC.

A **Slowsync** occurs when the prime condition for Fastsync is not met - when the PDA is syncing with a PC different from the one to which the it synced the last time Hotsync was invoked. In this case the PDA transfers the entire database to the PC to be worked upon by comparison. As PDA storage space becomes bigger and databases become larger, the issue of increasing latency involved in transferring the entire database from the PDA to the PC for subsequent synchronization becomes increasingly important. Figure 2.3 shows the difference in the communication complexity of Slowsync and Fastsync. Here, a database of appointments held on a PC and PDA is being synchronized using Slowsync and Fastsync. In the former protocol the PDA transfers its entire database to the PC and then the PC determines how to synchronize the databases while in the latter only the records which have been modified or added since the last synchronization are sent to the PC by the PDA.

It may be argued that with the advent of faster connection technologies for connecting mobile devices to PCs and networks and with the increasingly fast wireless LANs at hand, Slowsync latency becomes a tractable bottle-neck. Figure 2.5 shows some of the more recent technologies used by mobile devices to transfer data to and from PCs and networks, with a few representative PDAs and their approximate data storage capabilities marked out for comparison. It is interesting to note that even the state-of-the-art Firewire (IEEE 1394) [17] and USB [18] technologies are not able to solve the data transfer latency problem because of a corresponding increase in the amount of data being handled by newer devices. For example, a Palm Pilot (1MB

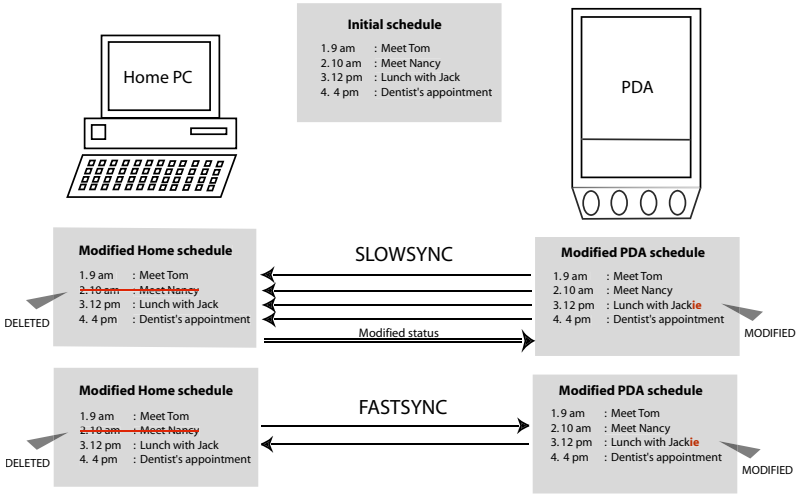


Figure 2.4: Difference between the two modes of the HotSync protocol. In Slowsync all data is transferred, while in Fastsync only modifications are sent.

storage) using a serial link to transfer its entire data to a PC (as is the case in Slow Sync) will take approximately the same order of time (73 seconds) as an Apple Ipod (5GB storage) using the Firewire 1394b data transfer technology (54 seconds).

2.2.2 Intellisync

The Intellisync Anywhere [19] product family from Pumatech always makes synchronization 'Fastsync enabled' by using one central server to which mobile devices always synchronize. This means that all synchronizing devices have to maintain status flags of updates with respect to only the central server, allowing for a Fastsync every time. Figure 2.6 shows a typical company network using the Intellisync architecture to setup and maintain a mobile device network for a Microsoft Exchange server. Information held at the exchange server is used to update information held on the Intellisync server. In addition, changes made by end users are stored on the Intellisync server and periodically synchronized with the exchange server. This periodic synchronization will result in a delay in the time it takes for updates made on

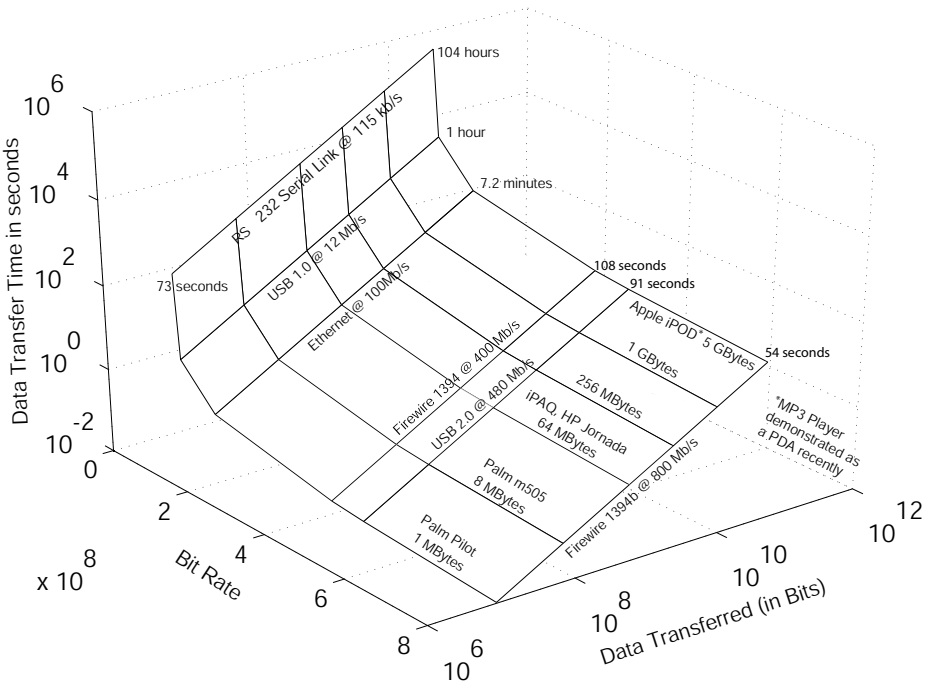


Figure 2.5: Recent faster data transfer technologies like USB and Firewire have reduced data transfer times by orders of magnitudes, though the potential size of data that may be transferred to or from a PDA has also increased accordingly.

the Exchange server to be available to users.

The centralized approach creates a central point of failure. If the central server is congested, the entire network suffers. In addition, scalability with the number of devices in the network is a serious issue because the load on the server increases linearly with increasing devices in the network. On the upside, there are many secondary benefits of a centralized scheme, including the feasibility of centralized security policy and easier methods of broadcasting information in an organization's network (such as policy changes, event schedules etc.).

The EDISON architecture proposed in [20] also relies on a centralized, shared

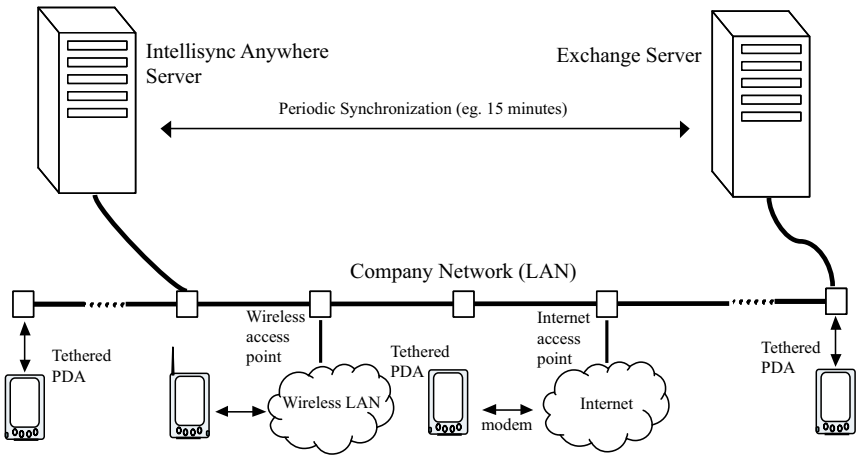


Figure 2.6: Intellisync Anywhere server installed on a company's network

server with which all hosts synchronize. The server maintains an incremental log of updates so that the hosts can always use Fastsync instead of Slowsync (see Section 2.2.1). 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 [21–23].

2.2.3 SyncML

SyncML [24] is an open industry initiative supported by many companies including Ericsson, IBM, Lotus, Matsushita, Motorola, Nokia, Openwave and Starfish. It seeks to provide an open standard for synchronization across various platforms and devices.

SyncML tries to achieve "Fastsync" capabilities by requiring that every device in the network keep status flags for each record with respect to every other device in the network. While this does guarantee the near optimal communication complexity for

synchronization, there is an unreasonable demand on the mobile device's memory to store the status flags of each record with respect to each other device in the network, making the memory complexity on each device O(nr) where there are n devices in the network and r records on the device. This is not scalable for multi-device networks with a large number of n devices. For example, there might be 100 devices synchronizing among each other and each of these hold $10,000$ records. If it takes 8 bytes to store status information for a record, each device will have to use approximately 8MB of memory to store status flags.

Version vectors [25] may be used as an approach to implement SyncML efficiently, which make the overhead linear in the number of updates m and not on the number of items in the data sets. It is not clear whether this approach will scale with the number of devices in a network because the authors in [25] assumed that the network size is small enough to be considered a constant. This assumption may not be applicable in a larger setting such as the one shown in Figure 2.6. The overhead would more accurately be $O(n \cdot m)$ where n is the number of devices in the network.

2.2.4 CPISync

There are many instances of data synchronization when there is a need to synchronize very similar data present on different hosts. CPISync [13, 26] makes the time complexity of this synchronization almost independent of the size of the data sets being synchronized and only dependent on the number of differences between the two data sets, thus approaching the lower information theoretic bounds of Equation 2.1.

This is in marked contrast to Slowsync which utilizes a wholesale data transfer between hosts while synchronizing data, making data synchronization time linear in

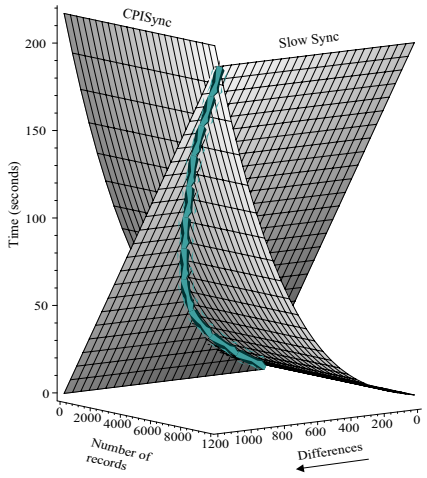


Figure 2.7: CPISync scales perfectly with increasing number of records in the database, while the Slowsync time increases linearly with growing number of records. CPISync time grows quickly with increasing differences between the synchronizing databases.

the number of records in the database.

CPISync, which is based on a recent solution to the set reconciliation problem [14] by Minsky, Trachtenberg and Zippel is scalable both with the number of devices in the network and the size of the databases being synchronized. Though this solution involves more computation as compared to Slowsync, the computation can be distributed asymmetrically between the hosts. This means that in case there are hosts with different computational capabilities (e.g. PC and a PDA), CPISync can transfer most of the computation to the PC, thus allowing for quicker execution of CPISync as compared to a symmetric approach.

Figure 2.7 shows the times taken by CPISync and Slowsync to synchronize data sets of varying sizes and differences on a PC-PDA synchronization system. The time complexity is the sum of the communication time of transferring data between the two hosts and the computational complexity of synchronization for CPISync and Slowsync. Slowsync is completely unscalable with increasing data set sizes due to

its linear dependence on the size of data sets being synchronized unlike CPISync which takes constant time for a given number of differences irrespective of the data set sizes.

In a PC-PDA synchronization setting the number of differences between successive synchronizations usually does not exceed a few. For example, in an address book of 500 addresses, it is unlikely that a user will add or more than 20 new addresses in a week. In such instances, the case for using CPISync is particularly strong, since for a small number of differences, the CPISync algorithm outperforms Slowsync significantly.

We discuss the CPISync algorithm in detail in Chapter 3.

2.3 Putting it together

Each of the synchronization protocols described above scale in some respects while not in others. In order to highlight these differences between them Figure 2.8 shows a set of radar diagrams that highlight the scalability characteristics of the protocols qualitatively. Each protocol is evaluated according to the following criteria:

Communication - This refers to the amount of data exchanged between the synchronizing devices. Communication is usually important because of two reasons. First, it directly affects the time needed for synchronization and secondly it has a bearing on resources used while synchronizing such as the battery consumed in mobile devices. Both these factors directly affect the user. Slowsync for example transfers the entire database from the PDA to the PC for synchronization and scores poorly on this count as shown in Figure 2.8.

Network size - The network size is the number of devices on the network that can synchronize. That is, the scalability of the protocol with the number of

devices that synchronize with each other. The point to note about any synchronization protocol is the amount of information it has to store to synchronize with each device on the network. SyncML requires too much memory to store synchronization information. Fastsync does not allow multi-device synchronization.

Robustness - The robustness here is characterized by the nature of the protocol itself - whether it is resilient to device failure in the network. Intellisync scales poorly because there is a central point of failure - a central intellisync server (see Figure 2.6) in the network whose failure will lead to disruption of the whole synchronization service for every device on the network.

Computation - This is the amount of computation the devices need to do in order to synchronize. Mobile devices and PDAs generally have weak processors (though this is changing). CPISync is computationally intensive, though most of the computation can be transferred to the PC, saving the PDA slow and intensive number crunching.

Memory - Memory is a precious resource on PDAs. Though the amount of memory on newer PDAs is rapidly increasingly, PDA memory is shared between the data and RAM. There is no separate secondary storage or "hard disk" to store data. A synchronization protocol that takes up minimum real estate in the PDA memory is always a better choice. SyncML scores poorly on this count because it requires each device to keep status flags for every other device in the network.

The ideal protocol with respect to the above criteria will correspond to the 'full' pentagon as shown in Figure 2.2. It can possibly achieved by an amalgamation of the desirable characteristics of the protocols discussed in the previous section.

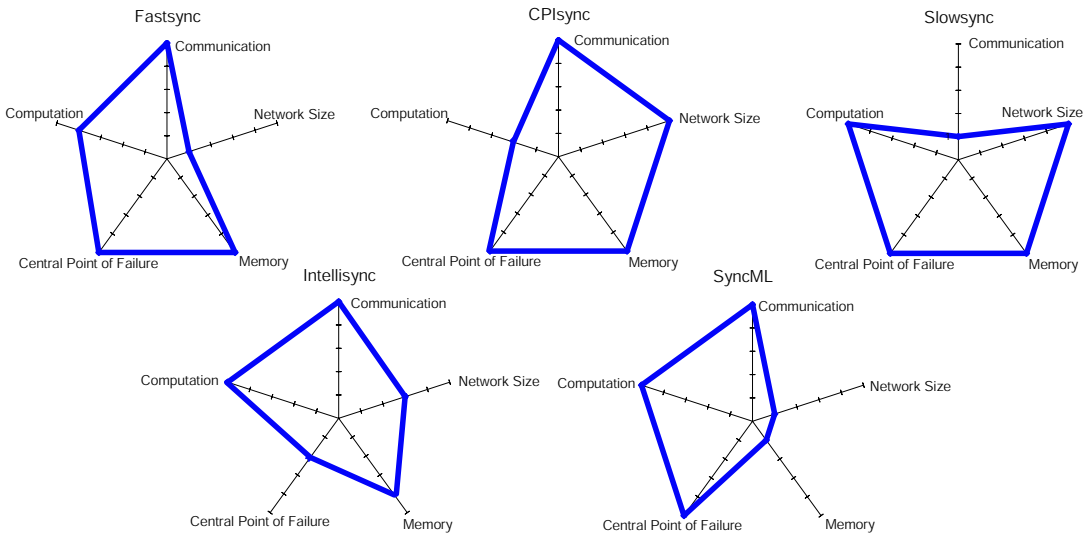


Figure 2.8: Scalability strengths and weaknesses of different synchronization protocols. An absolute qualitative scale is used in which high values indicate strengths and low values indicate weakness.

Chapter 3

PDA Synchronization

Personal digital assistants or PDAs are periodically linked up to computers in order to synchronize data being held on the PDA and the data held by complementary applications on the computer. A secondary benefit of periodic synchronization is that it serves as a backup mechanism. Typical application data that needs to be synchronized includes address books, memo pads, to-do lists, email clients, schedule planners and off-line internet and web information (like avantgo [27]). Information in these applications is modified on the PDA and/or the PC independently. The more often synchronization is done, more current is the data being held on the PDA and the PC. A natural result of frequent synchronization is that the amount of 'new' information between two successive synchronizations is very small compared to the total amount of information in the databases of the PDA and the PC.

In this chapter we first introduce CPISync (characteristic polynomial interpolation based synchronization) in Section 3.1 that is based on a set reconciliation algorithm first proposed in [14]. We then describe two variants of the original algorithm, namely Deterministic CPISync and Probabilistic CPISync in Sections 3.2 and 3.3. The algorithms' theory as well as implementation and results are described

in these sections.

The CPISync algorithm is suitable for application in PDA synchronization because it makes the communication and time complexity of synchronization a function of the differences between two databases being compared and not the size of the databases themselves. In this chapter we first discuss the CPISync algorithm and then go on to discuss its application in the PC-PDA synchronization system. Towards the end of the chapter we discuss the our results of the CPISync PC-PDA synchronization as compared to Slowsync, discussed in Section 2.2.1.

3.1 CPISync

Set reconciliation synchronizes unordered data sets, that is to say that the order of information in the data sets being synchronized is not important – such as an appointment schedule, where the data can be easily re-ordered later according to the time of appointments (say). There are two approaches to CPISync possible, one when an upper bound on the number of differences $\overline{m}$ between the synchronizing data sets is known and the other when $\overline{m}$ is not known and has to be intelligently *guessed*. We call the former approach the "Deterministic approach" and the latter "Probabilistic approach". In the Probabilistic approach we iterate the Deterministic approach repeatedly by increasing $\overline{m}$ in each iteration until a satisfactory reconciliation results.

3.2 Deterministic CPISync

Synchronization is done on two sets of *similar* data sets held on two different devices. By *similar* we mean that most of the records in the data sets are identical. Three types of updates may occur in databases - an addition of a record, a deletion of a record or a modification of a record, each introducing a *difference*.

Consider a set S_{PDA}, a data set on the PDA and a similar set on the PC, namely S_{PC}. S_{PDA} and S_{PC} were probably derived from the same identical set (a previous synchronization and subsequent independent updates on the hosts for example). For simplicity we consider sets of integers. For example $S_{PDA} = \{1, 2, 4, 5\}$ and $S_{PC} = \{5, 1, 6\}$. The objective of an efficient synchronization routine is to determine the set $S = S_{PDA} \bigcup S_{PC} = \{1, 2, 4, 5, 6\}$ and install this set S in the PDA and the PC in the least possible time, using the least amount of communication and computation.

CPISync first constructs a characteristic polynomial for each of the data sets it is reconciling which is simply a univariate polynomial whose factors are the elements in of the data set. These characteristic polynomials is sampled at $\overline{m}$ sample points. The PDA sends these evaluations to the PC which calculates rational function evaluations by taking the ratios of the evaluations of the characteristic polynomials at the same sample points and interpolating a rational function $f(z)$ fitting the points. The factors of the numerator and denominator of $f(z)$ give the different (missing) set data.

We now explain the CPISync Algorithm in detail by means of the example given above. Figure 3.1 outlines the steps of the algorithm which are itemized below.

1. The PDA calculates the characteristic polynomial function from its data set S_{PDA}. A characteristic polynomial function has all the data set elements as its factors, as is shown in figure 3.1. For example, for $S_{PDA} = S_A = \{1, 2, 4, 5\}$, the characteristic polynomial function $\chi_{PDA}(X) = (X-1)\cdot(X-2)\cdot(X-4)\cdot(X-5)$. This is the 'signature' of the information contained in the data set. The PC also calculates the characteristic polynomial function for its own data set $S_{PC} = S_B = \{5, 1, 6\}$; $\chi_{PC}(X) = (X-5)\cdot(X-1)\cdot(X-6)$. Note that the characteristic polynomials are always monic, i.e., the highest degree term of the polynomial

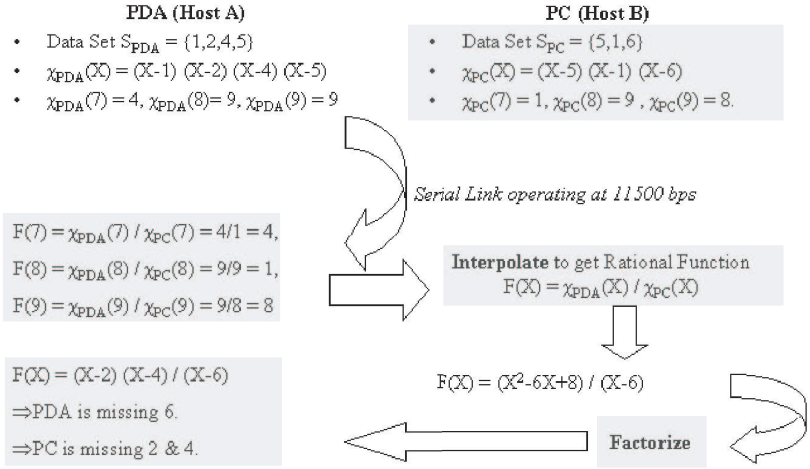


Figure 3.1: CPISync Algorithm - a simple example showing interpolation based synchronization

has coefficient 1.

2. The characteristic function on the PDA is then 'sampled' at some pre-selected random sample points. In our example, these are $\{7, 8, 9\}$. The number of sample points has to be at least equal to the number of differences in the system which explains the need to know an upper bound $\overline{m}$ on the number of differences in the system.

3. The sampled characteristic polynomial function values of the PDA are transferred to the PC from the PDA. This almost completes the PDA's job, it is now up to the computationally stronger PC to do the remaining steps of the reconciliation synchronization. In Figure 3.1 the shaded text indicates the computation performed on the PC: this is the asymmetric capability of the CPISync algorithm.

4. The PC calculates rational function instantiations by dividing each sample point evaluation of the PDA's characteristic polynomial function with its own characteristic polynomial function's evaluation at that same sample point, ending up with *2-tuples* of *(sample point, rational function evaluation)*.

5. These *2-tuples* are then interpolated to obtain the rational function $F(X)$. This rational function is of the type $\frac{Pu(X)}{Qu(X)}$ where $Pu(X)$ and $Qu(X)$ are (unreduced) polynomials with coefficients in the finite field F_{11} (in our example). $Pu(X)$ and $Qu(X)$ are divided with their GCD to obtain an irreducible polynomial of the form $\frac{P(X)}{Q(X)}$. The sum of the degrees of $P(X)$ and $Q(X)$ equal the total number of differences between the data sets S_{PDA} and S_{PC}.

6. $P(X)$ and $Q(X)$ are separately factored yielding the differences the PDA and the PC are missing respectively. The factors of $P(X)$ are sent to the PDA to update its data set, while the factors of $Q(X)$ are used by the PC to update its own data set and incorporate the new data from the PDA that it had been missing.

   Note that the last step is the most simplistic scenario. There might be instances of conflicts (see section 2.1) which would need to be dealt with separately.

There are some subtleties that need to be explained at this point. The first is the use of finite field arithmetic to limit the size of the characteristic polynomial evaluations because their calculation involves evaluating a product of factors which grows very rapidly. We have used F_{32749} for the PC-PDA synchronization, which gives us a 15-bit space for the mathematical operations. One bit out of these is used for the sample points, which are discussed next.

There are two things that need to be explained about the choice of the sample points. The sample points need to be separate from the data points in the set. This

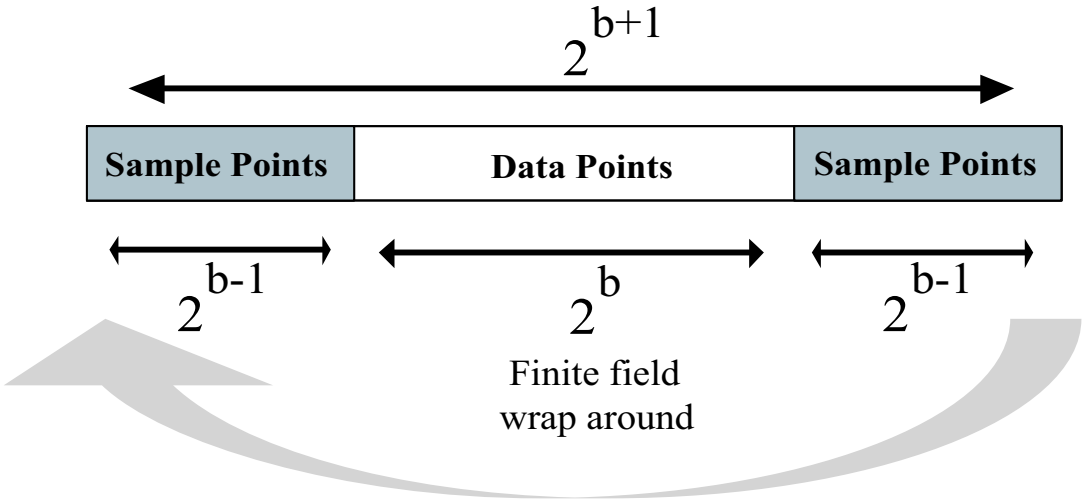


Figure 3.2: The finite field space is divided into two regions, one for sample points and the other for data points

is because otherwise, the characteristic polynomial of the corresponding data set will evaluate to 0 in step 2. In our example (Figure 3.1), if we selected 2 as a sample point, the characteristic polynomial evaluation $\chi_{PDA}(2) = (2-1)\cdot(2-2)\cdot(2-4)\cdot(2-5) = 0$. A bad choice may result in a division-by-zero error in step 4 of the algorithm.

Since we do not know a priori the number of differences in the data to be reconciled, we increase the size of the finite field by approximately 1 bit. One bit provides adequate space to accommodate the sample points because for b bit data, there can be a maximum of 2^b differences between any two data sets and CPISync requires at least as many sample points as there are differences. This concept is shown in figure 3.2. The idea here is to pick the sample points from a separate 'region' of the *number line* than the data points, eliminating the possibility of a 'collision' between the data and sample points.

The specific choice of the region of sample points in Figure 3.2 is because choosing sample points symmetrically over the 0 point speeds up calculations in the Gaussian elimination involved in interpolation in step 2 of the algorithm described above.

Symmetric sample points (such as $0, 1, -1, 2, -2...$) help in making the Gaussian elimination faster by making it simple to get many zeros in the matrix while solving for the coefficients of the rational function, thus speeding up Gaussian elimination. This is particularly important because as we will see later, most of the latency arises from this Gaussian elimination. Note that exact symmetry about 0 is possible only for an odd number of sample points.

A second point of interest to mention here is the choice of the finite field. We have used F_p where p is a prime number. For the Palm PDA, the word size is 16 bits. We carefully chose $p = 32749 (= 2^{15} - 19)$. This is the largest prime number less than $2^{15} (= 32768)$.

In case the records are variable length strings, they may be hashed into fixed small length hashes to be used as the data set elements that are reconciled using CPISync. The difference information in terms of hashes reported by CPISync may then be transformed into information about the corresponding differing strings using a reverse table lookup. We have avoided the issues of hashing by restricting our record database entries to 15-bit integers. We note that, in practice, the hashing operation needs to be performed only once per entry, at the time that the entry is added to the data set, thus the complexity of the hashing is not a bottleneck for synchronization. By restricting entries to 15 bits, we have also avoided the issues of multiple-precision arithmetic on the PDA.

Figure 3.3 shows the high level scheme of the PDA-PC synchronization. The thing to note here is that the transfer of data (indicated by the arrows) is typically through a slow link. In the case of a Palm PDA, the transfer is through a serial cable or IrDA (Async Serial-IR 9600-115.2kb/s) [28]. Newer PDAs have USB [18] and Ethernet connectivity. These technologies are substantially faster than serial links, though newer PDAs often run more data intensive applications, leaving the problem

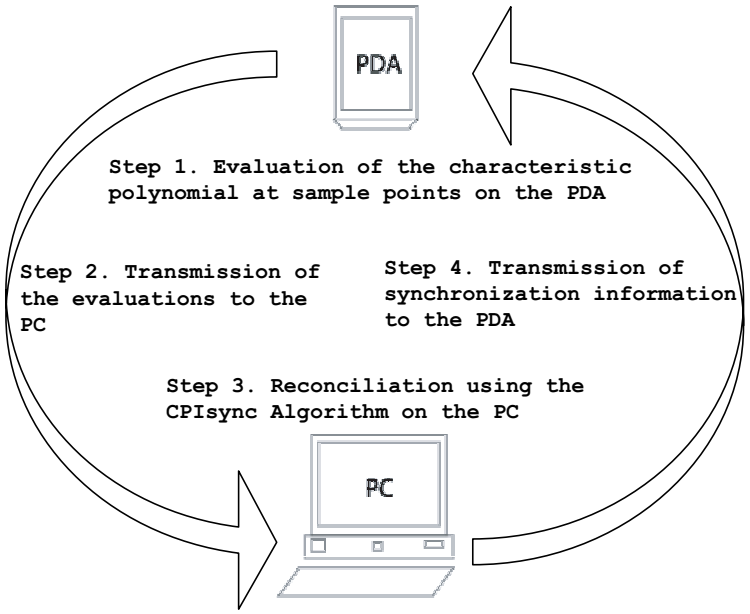


Figure 3.3: The overall scheme of the PC-PDA synchronization system

of slow data transfer between the PDA and the PC network intact, as was shown in Figure 2.5.

Figure 3.3 indicates that the total communication between the PDA and the PC is simply the transmission of the characteristic polynomial function evaluations from the PDA to the PC (there are $\overline{m}$ such evaluations). After the reconciliation is complete, the number of data set elements the PDA is missing ($\overline{m}$ at worst) are transferred back to the PDA. Thus taken totally, the number of data elements transferred is O($\overline{m}$), irrespective of the sizes of the data sets on the PDA and the PC.

System Model

The PC-PDA model used a Palm IIIxe with a 16 bit Motorola Dragon ball processor [29] and 8MB of RAM. The palm was synchronized with a Pentium III class machine with 512 MB of RAM. We used a serial RS-232 link operating at 115$kbps$ as the link between the PDA and the PC. Our specific implementation emulates a

memo pad implementation. As data is entered into the palm, evaluations of the characteristic polynomial (described in the previous section) are updated at designated sample points. Upon a request for synchronization, the Palm sends $\overline{m}$ of these evaluations to the PC, corresponding to the differences between the data and the two machines. The desktop compares these evaluations to its own evaluations and determines the differences between the two machines, as described in section 3.2 on page 23.

We do not address issues about which specific data to keep at the end of the synchronization cycle in case of conflicts, but several techniques from database literature explained in [30] are possible candidates for conflict resolution.

Finite field arithmetic was performed with Victor Shoup's Number Theory Library and data transferred between the PDA and the PC in *pdb* (Palm database) format. This data was converted to text format using [31]. This data is converted to input suitable to the CPISync system on the PC using a separate Java language based string parser. Figure 3.4 shows a sample script of an experimental run of our system and explains how the PC-PDA system is setup. Our model uses the Hotsync middleware of the Palm operating system to transfer data as *pdb* files. We have used the default conduit provided in the Palm OS API [32] for this purpose.

In the experiments we performed, the number of differences between the data sets held on the PC and the PDA varied from 0 to about 1200 and the size of the data sets varied from 100 to 10000. Since it was not practical to manually introduce these ranges of data and differences, we implemented an automatic data generator on the PC and the PDA. The program on the PDA was called *Reconciliation Genie*. This program took two inputs – the size of the database and the number of differences to be introduced. Instead of actually generating this data set, *genie* just calculated the characteristic polynomial of the data set elements (they were serially ordered).

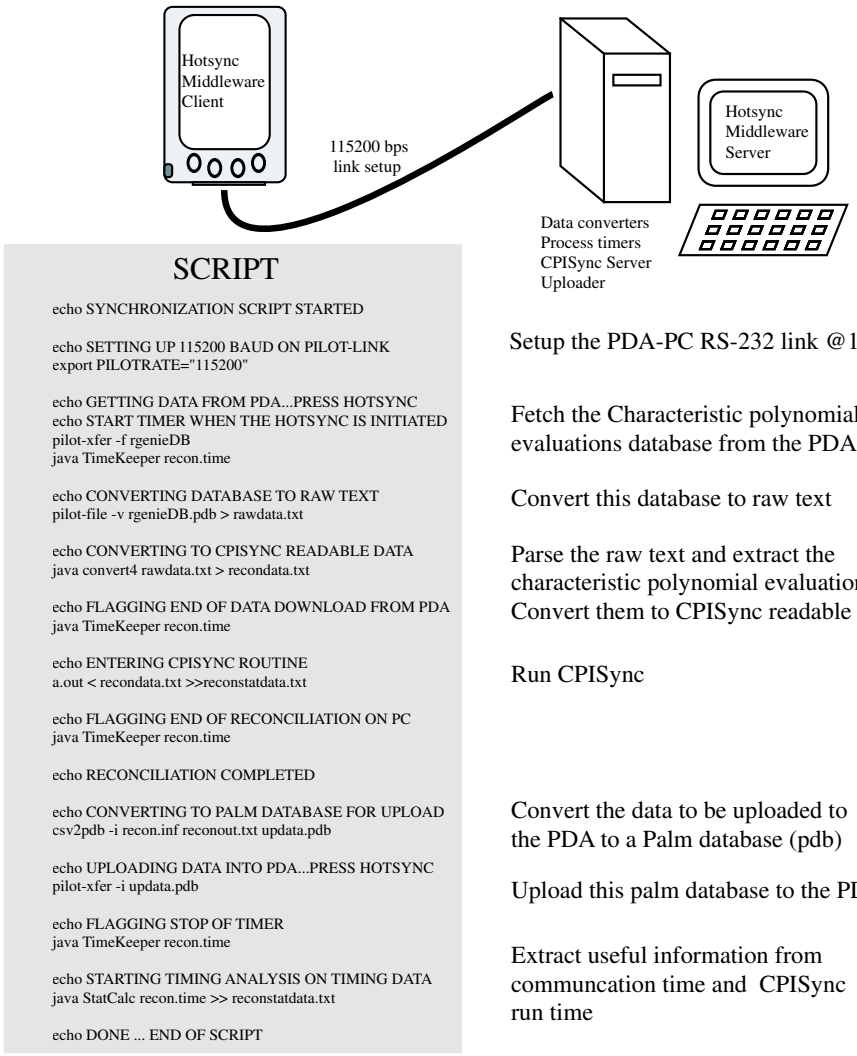
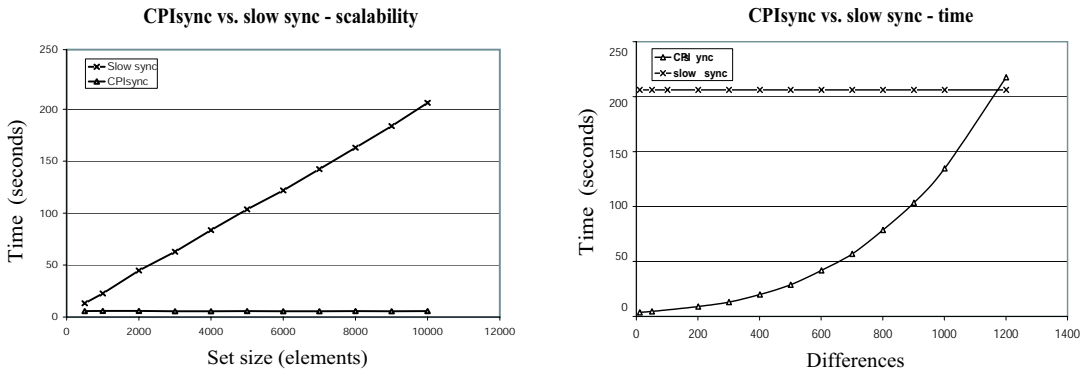


Figure 3.4: A sample script to illustrate an experiment on the PC-PDA system

The range of the data elements could be between 8188 and 24561 in the finite field F_{32749}. This particular choice is because out of the total 32749 elements in the finite field, the first and last quarter are reserved for sample points (as was explained in section 3.2.

A complementary program on the PC took the same inputs – the size of the database and the number of differences to be introduced and constructed a database accordingly. So for testing purposes, we simulated a situation wherein we 'knew'



(a) (b)

Figure 3.5: Comparison between (a) Scalability with the size of the data sets (b) the time complexities of CPISync and Slowsync for increasing differences.

the number of differences so as to set up the desired synchronization set size and difference parameters. It is important to note here that the CPISync system on the PDA and the PC itself was not a simulation.

3.2.1 Metrics and Measurements

We consider the time required for a synchronization to complete as the most important metric in our experiments. There are two factors that jointly contribute to the total synchronization time - the data transfer time and the computation time to synchronize. For Slowsync the dominant component of the latency is the data transfer time, whereas for CPISync the computation time generally dominates. Our experiments compare the latencies of CPISync and Slowsync in various scenarios. The synchronization latency is measured from the time at which the PDA beings to send its data to the PC until the time at which the PC determines the differences between the databases. On the down link from the PC to the PDA, both Slowsync and CPISync will transmit the same updates.

3.2.2 Results

We ran the CPISync and Slowsync protocols on the PC-PDA synchronization system for various set sizes - varying from sets of 100 elements to sets of 10000 elements with differences varying from 0 to more than 1200 for these data sets. We timed the total latency of the synchronization as the sum of the communication time (time taken to transfer data from the PDA to the PC) and the computational time (time taken to reconcile the data sets), ignoring the time to upload data from the PC back to the PDA since this is identical for both the protocols. Figures 3.6 and 3.7 show some of the data we collected. The values given are averages over 10 trials of identical experiments.

Figure 3.5 (a) depicts the superior scalability of CPISync over Slowsync. In this figure we have plotted the time used by each synchronization scheme as a function of data set size for a fixed number of differences between data sets.

It is clear from the resulting graphs that Slowsync is markedly non scalable: the time taken by Slowsync increases linearly with the size of the data sets. CPISync on the other hand is almost independent of the data set sizes. Comparing Figure 2.3 to Figure 3.5 (a) we observe that the qualitative behavior of CPISync is similar to that of Fastsync. The remarkable property of CPISync is that it can be employed for any synchronization scenario, regardless of context, whereas Fastsync is employed only when the synchronization took place between the same PC and the same PDA.

In figure 3.5 (b), we compare the performance of CPISync to Slowsync for data sets with fixed sizes but increasing number of differences. As expected, CPISync performs significantly better than Slowsync when the two synchronizing data sets do not differ by much. However as the number of differences between the two data sets increase, the computational complexity of CPISync becomes significant. Thus,

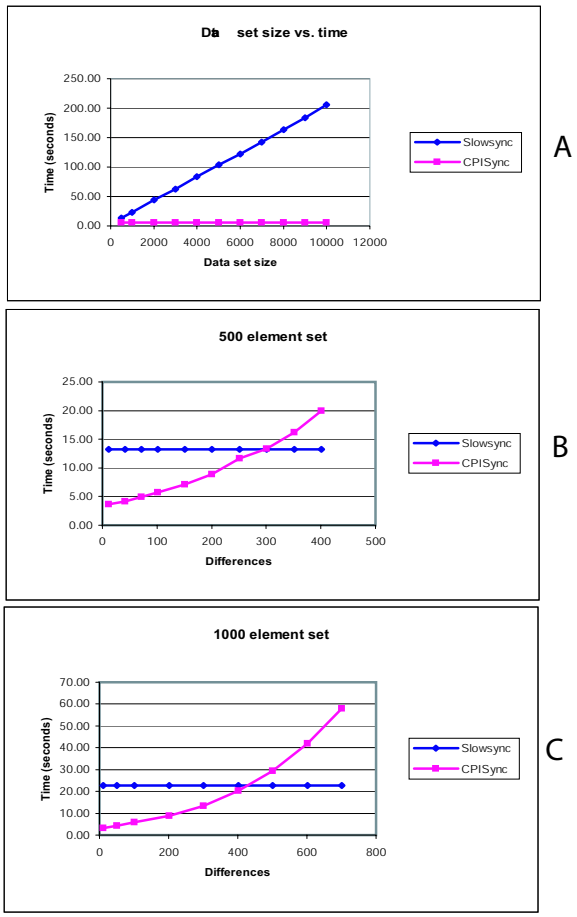


Figure 3.6: (A) highlights the superiority of CPISync over Slowsync by showing that Slowsync does not scale with increasing number of elements in a data set and a fixed number of differences. The next two graphs show the comparison the latency of CPISync and Slowsync for (B) 500 and (C) 1000 element sets and varying differences.

there exists a threshold where wholesale data transfer (Slowsync) 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 $10,000$ record sets depicted in the figure, this threshold corresponds to roughly 1200 differences.

By preparing graphs like Figures 3.6 and 3.7 for various different set sizes, we are able to produce a regression with coefficient of determination[1] [33] almost 1

[1]This is a measure of how good the fitted curve is - it is determined by normalizing the square of the difference between the given points to which the curve is fitted and the evaluation of the

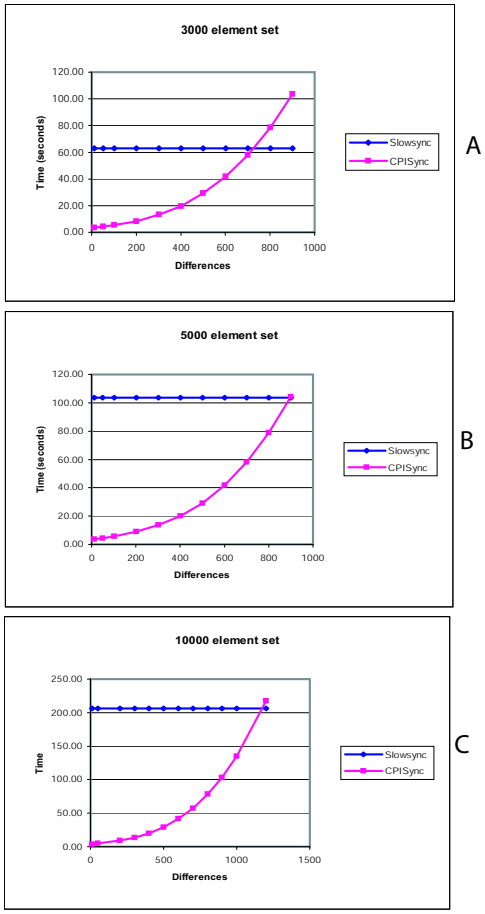


Figure 3.7: These three graphs show the comparison the latency of CPISync and Slowsync for (A) 3000, (B) 5000 and (C) 10000 element sets and varying differences.

that analytically models the performance of Slowsync and CPISync. The resulting threshold values are listed in Table 3.1. The regression for Slowsync is obtained *using Maple* by fitting the data to a linear function that depends only on the data set size, where as for CPISync the regression is obtained by fitting the data to a cubic polynomial *using Maple* that depends only on the number of differences. Note that in a PDA application like an address book or memo, changes between concurrent synchronization typically involve only a small number of records. For such applications, CPISync will usually be much faster than Slow Sync.

curve's interpolated function at the corresponding value

In Table 3.2, we show the fitted cubic polynomials generated using *Maple* corresponding to 100, 250, 500, 1000, 3000, 5000 and 10000 element data sets for the total time(latency) with parameter m (differences). It is interesting to note that for a small data sets (and consequently small differences), it is the communication time which seems to dominate and the cubic fit is not good, whereas for sets of size 1000 or more, the cubic computational complexity dominates the total latency. The communication time is dependent on the operating systems on the PDA and the PC and there was a high level of inconsistency observed in the timing (of the order of $\pm 30\%$) even though all these results have been averaged out over 10 trials. In the table, we see there is an overhead of about 3.3 seconds (constant term coefficient) which corresponds to the setup time of the protocol and other operating system overheads.

For bigger size sets ($\geq$ 1000), it is interesting to note from Table 3.2 that the highest order (cubic) coefficient is almost the same irrespective of the number of elements in the set. This again shows that CPISync is not dependent on the size of the data set. In fact, for big data sets with considerable differences, the latency may be represented by any one of the polynomials in Table 3.2, since for large values of m the cubic term dominates the value of the latency. We shall use this fact to model an optimal Probabilistic scheme in section 3.3.

Data set Size	*Differences*
100	161
250	175
500	253
1000	431
2500	620
3000	727
5000	899
10000	1177

Table 3.1: Threshold values at which CPISync takes the same time as Slowsync.

The implementation of CPISync described here requires the knowledge of a tight upper bound $\overline{m}$, on the number of differing entries. One simple method for obtaining such a bound involves having both hosts A and B count the number of modifications to their data sets since their last synchronization. The next time the hosts synchronize, host A sends to host B a message containing its number of updates, denoted $\overline{m}_A$. Host B uses its own value $\overline{m}_B$ to compute $\overline{m} = \overline{m}_A + \overline{m}_B$ which is the upper bound on the total number of differences between the two hosts. Clearly, this bound $\overline{m}$ will be tight if the two hosts have performed updates to mutually exclusive records. However, they may be completely off if the hosts have performed exactly the same updates to the same records in their databases. This may happen if, prior to synchronization, both hosts A and B synchronized with a third host C. 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 will be rather ineffective for some applications. In the next section we describe a probabilistic scheme that can determine a much tighter value of $\overline{m}$. This result is of fundamental importance as it means that, in a general setting, both the communication and computational complexities of CPISync depend mostly on $\overline{m}$.

Data set Size	*Fitted Cubic Polynomial*
100	$-3.18573E-06m^3 + 5.41161E-04m^2 + 2.66366E-03m + 3.33124$
250	$1.05747E-06m^3 - 2.45622E-04m^2 + 3.36804E-02m + 3.23576$
500	$6.28175E-08m^3 + 2.70934E-05m^2 + 2.02042E-02m + 3.33711$
1000	$1.01182E-07m^3 + 7.74547E-06m^2 + 2.32522E-02m + 3.13590$
3000	$8.79353E-08m^3 + 2.59495E-05m^2 + 1.63701E-02m + 3.45446$
5000	$9.76246E-08m^3 + 1.37409E-05m^2 + 2.02848E-02m + 3.43659$
10000	$1.03861E-07m^3 + 4.79533E-06m^2 + 2.26613E-02m + 3.40529$

Table 3.2: Fitted polynomials $P(m)$ for CPISync running time for various data set sizes.

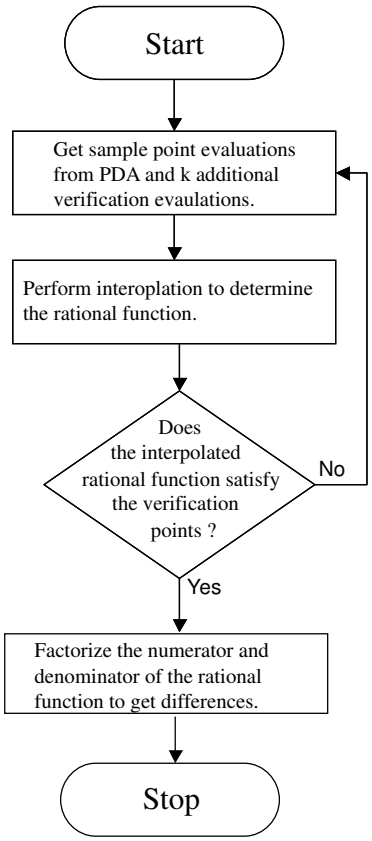


Figure 3.8: Probabilistic scheme

3.3 Probabilistic CPISync

An extension of CPISync when the upper bound on the number of differences between the reconciling hosts is not know is performed in the following manner.

- 1. Hosts A and B guess an upper bound $\overline{m}$ and perform the deterministic CPISync described in the previous section, resulting in host B computing a rational function $F(X) = \frac{P(X)}{Q(X)}$ as described in 3.2 on page 23. If the function $F(X)$ corresponds to the differences between the two host sets, that is if

$$F(X) = \frac{\chi_A(X)}{\chi_B(X)} \tag{3.1}$$

then the zeros of $F(X)$ will determine precisely the mutual difference between the two sets. In order to check this, host A (PDA) chooses k random sample points r_i and sends their evaluations $\chi_A(r_i)$ to host B (PC), which uses these evaluations to compute evaluations

$$G(r_i) = \frac{\chi_A(r_i)}{\chi_B(r_i)} \tag{3.2}$$

by comparing the evaluations $F(r_i)$ and $G(r_i)$, the PC (host B) can assess whether F(X) is indeed the correct interpolation for the rational function $\frac{\chi_A(X)}{\chi_B(X)}$.

- 2. In case each $G(r_i)$ does not correspond to $F(r_i)$ for each r_i we repeat step 1 for a larger value of $\overline{m}$

In general, the two hosts keep guessing $\overline{m}$ until the resulting polynomials agree in all k sample points. A precise probabilistic analysis in [14] shows that such an agreement corresponds to a probability of error

$$\epsilon = m\left(\frac{|S_A| + |S_B| - 1}{s^b}\right)^k \tag{3.3}$$

where m is the number of differences between the two reconciling hosts A and B. Using a trivial upper bound $m \leq |S_A| + |S_B|$ we see that an arrangement of

$$k = \left\lceil log_\rho\left(\frac{\epsilon}{|S_A| + |S_B|}\right)\right\rceil \tag{3.4}$$

samples (where $\rho = \frac{|S_A|+|S_B|-1}{2^b}$) to get a probability of error ϵ for the whole protocol. Thus, for example reconciling host sets of 10^6, 64-bit integers with error probability $\epsilon = 10^{-20}$ would require agreement of $k = 2$ random samples.

This verification protocol requires the transmission of at most $m+k$ samples and one random number seed (for generating random sample points) to reconcile two sets; the value k is determined by the desired probability of error ϵ using the above expression for k. The m sample points are used to interpolate a rational function, corresponding to a guess of the differences between the two machines, and the latter k points are used to verify the correctness of this guess. If the verification succeeds, the synchronization is complete. On the other hand, if verification fails, then the PC collects all the sample points seen so far into a guess of the differences between the two machines at the same time requesting k additional random evaluations from the PDA to confirm its new guess. This procedure is iterated until the verification succeeds, at which point the synchronization is complete. Since m evaluations will necessarily be enough to completely determine up to m differences, verifications will necessarily succeed after at most $m+k$ transmissions. Figure 3.8 shows a flowchart of probabilistic CPISync.

Thus, though the verification protocol will require more rounds of communication for synchronization that the Deterministic CPISync, it will not require transmission of significantly more bits of information. We see in the next section that the computational overhead of this protocol is also not large.

3.3.1 System Model

From an end-user perspective, the most important metric of usability of any synchronization protocol is the latency in the synchronization process. We therefore consider optimizing the latency with the view of minimizing it. We thus propose a scheme whose completion time is at worst a constant α times larger than the time needed to synchronize the two hosts when the number of differences is know a priori (using the

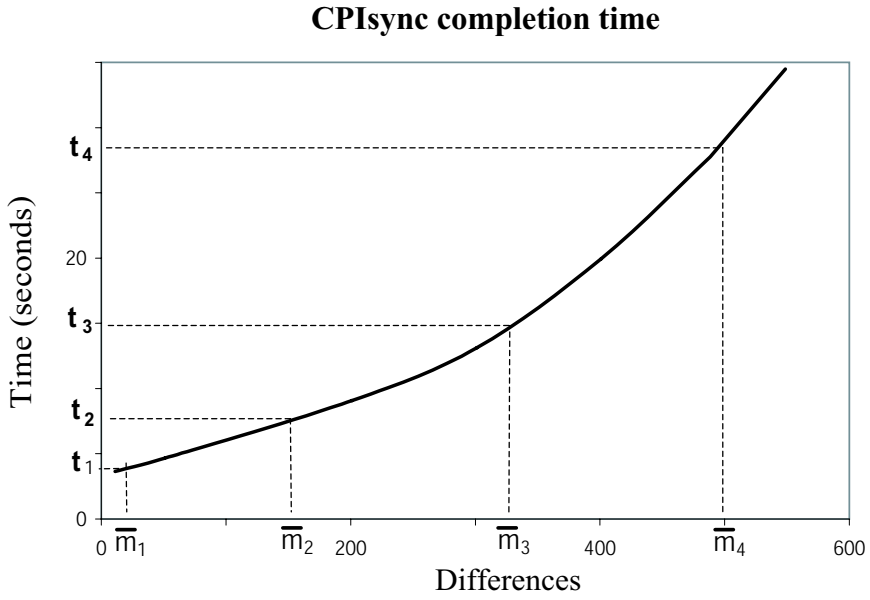


Figure 3.9: A model of the approach used to optimize the latency of synchronization when no bound is known on the number of differences between data sets.

Deterministic scheme discussed in 3.2 on page 23). This probabilistic scheme retains one of the essential properties of its deterministic counterpart: the synchronization latency depends only on the number of differences and not on the total size of the host data sets. We prove that $\alpha = 4$ is the optimal bound for this scheme and show how to achieve it.

Our approach to this optimization relies in part on the data graphed in Figure 2.7. We fit our data to a polynomial regression that interpolates the latency of CPISync as a function of the number of differences m between the two hosts as was shown in Table3.2. Since an exact value for m is not known at the start, the PDA and PC start with an initial guess $\overline{m}$ for an upper bound on m. In Figure 3.9, this initial guess corresponds to a value $\overline{m}_1$, which corresponds to a verification time of $t_1 = 3.65$ seconds. If verification fails this guess, we update our bound to the value $\overline{m}_2$ that

corresponds to a verification time that is δ times larger than for $\overline{m_1}$ differences(*i.e.* $t_2 = \delta \cdot t_1$). In the case of Figure 3.9, $\delta = 2$ giving $\overline{m} = 151$ and $t_2 \approx 7.29$ seconds. At each iteration we guess the bound $\overline{m}_i$ such that $t_i = \delta \cdot t_{i-1}$. We continue until verification succeeds for some guessed bound $\overline{m_n}$ requiring time $t_n = \delta^{n-1} \cdot t_1$ [13]:

Theorem 1 *The latency-optimizing probabilistic scheme takes at most* $\alpha(\delta) = \frac{\delta^2}{\delta - 1}$ *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 know a priori, and by $T(m)$ the synchronization latency required by this probabilistic scheme. Furthermore, denote by t_i the time needed for the i-th verification round in which $\overline{m}_i$ differences are guessed between the two hosts.

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

$$T(m) = t_1 + \ldots + t_n = t_1 + \delta \cdot t_1 + \ldots + \delta^{n-1} \cdot t_1 = \frac{\delta^n - 1}{\delta - 1} \cdot t_1 \qquad (3.5)$$

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

$$\frac{T}{T^*} \geq \frac{\delta^n - 1}{(\delta - 1)\delta^{n-2}}, \qquad \text{for all } n > 1. \qquad (3.6)$$

It is easy to check that the right hand side of (3.6) is maximized when $n \to \infty$, meaning that $T/T^* \geq \delta^2/(\delta - 1)$. By examining the derivative of $\alpha(\delta)$ with respect to δ, one finds that this function attains a minimum value at $\delta = 2$, leading to an optimal ratio of $\alpha = 4$. Thus, the best policy for this scheme is to double the

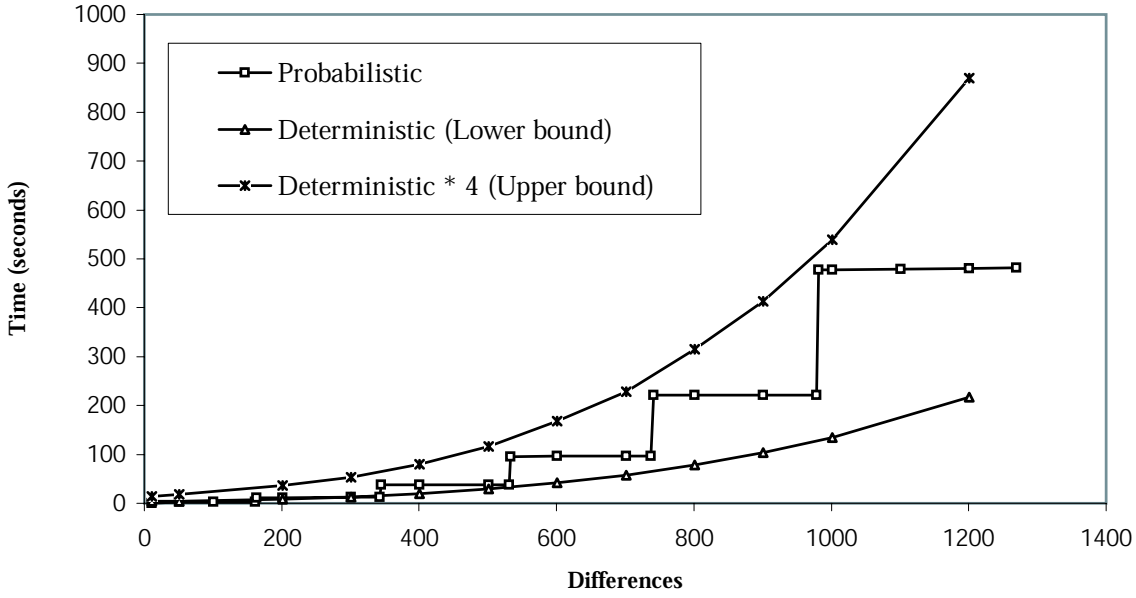


Figure 3.10: A comparison of the Probabilistic scheme with no known upper bound $\overline{m}$ to the deterministic scheme with a given value of m.

verification time at each iteration.

3.3.2 Results

Figure 3.10 illustrates the performance of this probabilistic scheme compared to that of the deterministic scheme. Note that the probabilistic results remain within the guaranteed factor 4 of the corresponding deterministic results.

It is interesting to note the 'staircase' pattern of Figure 3.10. The sharp inflection points corresponding to the 'steps' are the values of differences where the Probabilistic algorithm upgrades the value of $\overline{m}$. The computational time is almost totally dominated by the Gaussian elimination while interpolating (see [14] for details) – the vander monde matrix used has dimensions $\overline{m} * \overline{m}$ and the sharp inflection points

corresponds to the algorithm 'upgrading' $\overline{m}$ after an unsuccessful verification. The factoring of the polynomials is not the main computational bottleneck, even though in theory both Gaussian elimination and factoring are *cubic* in complexity.

Chapter 4

Conclusions

This thesis was divided into two major parts. In the first part, various synchronization technologies were investigated (Chapter 2) while in the second part the CPISync algorithm for fast peer-to-peer synchronization was discussed in detail (Chapter 3). Strengths and weaknesses of various synchronization technologies were highlighted and the compared to the new CPISync algorithm. The CPISync algorithm was shown to be more scalable as compared to whole data transfer algorithms like Slowsync.

4.1 Summary

Various synchronization protocols and technologies like Hotsync (Fastsync and Slowsync), Intellisync, SyncML and CPISync are suitable for specific classes of synchronization problems because they scale only with certain parameters and not with others. For example, the total time is determined by the communication complexity and corresponding time for sending synchronization data over the network connecting the synchronizing devices as well as the amount of time required for any computations while determining the differences. The memory used on each device to maintain synchronization information is also taken into consideration as one of the scalability

criteria, as is the number of devices that can synchronize among themselves (the size of the network). Finally, the robustness of the protocol - whether it is centralized or peer-to-peer is another scalability criteria.

Fastsync only works for two device networks while Slowsync does not scale with large data sets on the synchronizing hosts. Similarly Intellisync depends entirely on a central server for synchronization instead of a peer-to-peer approach which makes it prone to central point of failure problems. SyncML is not scalable with a large number of devices on the network because it requires each device to maintain synchronization information for each record with respect to every other device in the network. CPISync alleviates all these problems, but is expensive when there are many differences between the synchronizing hosts due to the computational complexity being cubic in the number of differences between the synchronizing hosts.

The CPISync algorithm has been implemented in a PC-PDA synchronization setting and has been shown to perform better than Slowsync for the case when the number of differences between the synchronizing PC and PDA is not overly large, as is the typical case in successive PC-PDA synchronizations. Since CPISync based Synchronization is independent of which particular device the PDA synchronized to the last time, the same algorithm can be used repeatedly while synchronizing a PDA with different devices, unlike Fastsync. On the memory front, the device has to maintain synchronization information which is of the order of $O(\overline{m})$, $\overline{m}$ being the upper bound on the number of differences between the two synchronizing hosts. The same information can be reused to synchronize with any number of devices.

The CPISync algorithm is thus able to solve the synchronization problem efficiently for many cases of synchronization, although as mentioned earlier, it will not work well for a large number of differences between synchronizing hosts. A more intelligent mechanism of choosing a synchronization protocol depending on the

specifics of the data being synchronized will solve the synchronization protocol most efficiently. For example, if the two devices synchronize successively then Fastsync may be employed, otherwise CPISync or Slowsync may be chosen depending upon the number of differences between the two devices.

4.2 Future Work

Many problems in networking arise from the need to periodically match up information held by hosts. Representative examples include OSPF link state routing [34, 35], Resource discovery in networks [10], Gossip protocols [11, 21, 36], QoS routing [37, 38] and Replicated database updating [25]. These applications often end up sending redundant information over the network in order to synchronize similar data held on hosts. This is one of the foremost scalability problems in these protocols - how to keep the protocol communication overheads low while guaranteeing accuracy and correctness.

The issue here then is to reduce the amount of redundant information being exchanged between hosts, which makes the CPISync algorithm an option while synchronizing host-to-host information because the number of differences between two successive versions of the information are typically very small as compared to the size of the information. For example, the number of changes in resources available on a network typically vary very slowly and so the change in information about resources is very small between updates in resource discovery. Attempting to employ CPISync for these problems is an exciting extension of CPISync to network synchronization problems.

As an initial incentive to explore these directions, we have developed a simple simulator called *Net sim* to simulate the amount of data transfer and the corresponding

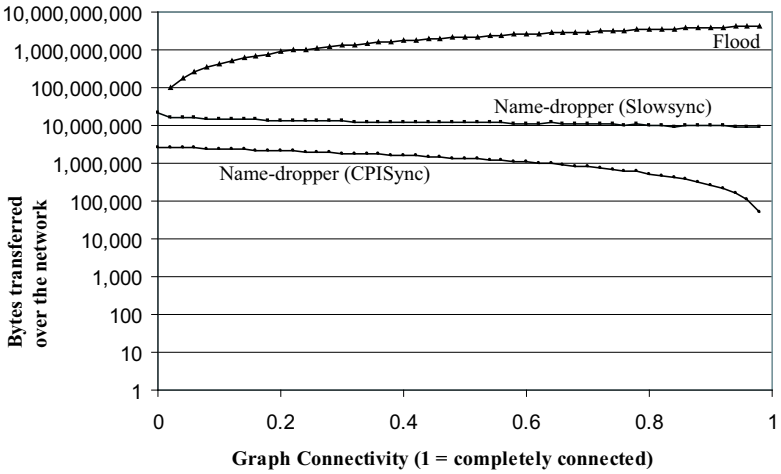


Figure 4.1: Simulated network usage in terms of bytes transferred for Flooding and Name-dropper resource discovery algorithms in a 1024 host network. The total bytes is the total number of bytes transferred over the network. Notice the superior performance of CPISync based Name-dropper as compared to Slowsync based Name-dropper

amount of time taken to run the Name-Dropper resource discovery algorithm [10] on a randomly generated network topology using CPISync and Slowsync as the underlying device-to-device synchronization methods.

The Name-Dropper resource discovery algorithm conveys information about resources available on a network to all the hosts connected to the network. In its simplest form, the Name-Dropper may be employed to enable hosts to discover other hosts on the network. Name-Dropper works like a gossip protocol in which hosts contact their neighbors randomly to get information about *their* neighbors and this continues until all the information is propagated throughout the network.

Figures 4.1 and 4.2 shows some of the preliminary results of our experiments. We simulated a 1024 node random network with increasing connectivity and determined the amount of communication (Figures 4.1) and the corresponding time(Figure 4.2) required to complete a Name-Dropper run. As expected, using CPISync to exchange

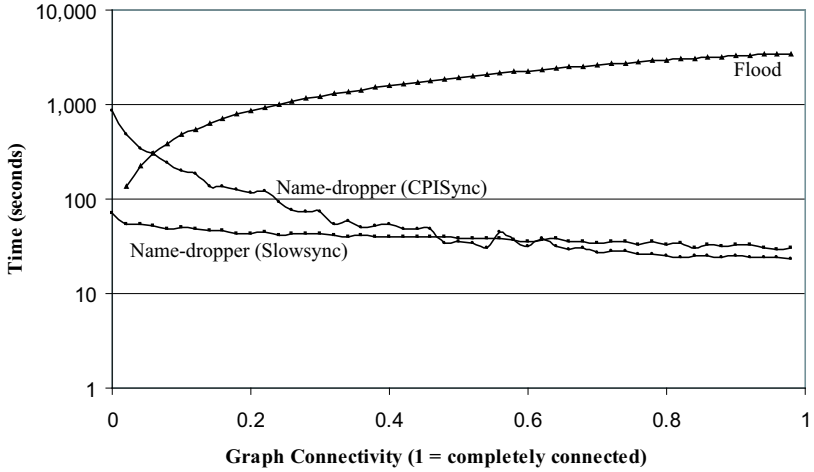


Figure 4.2: Simulated network usage in terms of time for Flooding and Name-dropper resource discovery algorithms in a 1024 host network. The total time is the sum of the computation time and the time taken to transfer the data, with the worst case bottleneck in the network operating at 56kbps. Notice that for well connected graphs, CPISync based Name-dropper is faster than Slowsync based Name-dropper.

information between adjacent hosts resulted in considerable gains.

These experiments point to the application of CPISync to the more general problem of device-to-device synchronization in heterogeneous distributed networks.

A more complete PC-PDA synchronization system that can generally synchronize PDA and PC databases is also an exciting extension to the work presented in this thesis. It would be particularly interesting to apply some heuristics to determine when the devices should switch between various modes of synchronization - Fastsync, Slowsync and CPISync for example.

Another interesting application field for the CPISync algorithm would be in the area of synchronization network algorithms [39] that seek to synchronize entire networks of devices by performing successive two-host synchronizations with the order being decided by how a data set containing the host ID might have been sorted using various classical sorting algorithms - the order of comparison of elements in the host

ID data sets corresponds to the order of synchronizing two hosts with those IDs.

The CPISync algorithm itself may be improved in ways to make the computational complexity more tangible for a large number of differences being reconciled. Recent work suggests that the cubic complexity (in the number of differences) of the algorithm may be made linear using a divide and conquer approach [40]. The biggest part of the latency in CPISync synchronization is the gaussian elimination step in order to interpolate the rational function as was discussed in section 3.2. Any improvements in this area will have a direct bearing on the efficiency of CPISync.

Bibliography

[1] C. E. Perkins, *Ad hoc Networking*. Addison-Wesley, first ed., 2000.

[2] "3COM Palm Inc.." http://www.palm.com.

[3] P. Mochapetris, "DNS: The Domain Name System," RFC 1034, Network Working Group, ISI, November 1987.

[4] P. Vixie, S. thomson, Y.Rekhter, and J. Bound, "Dynamic Updates in the Domain Name System (DNS UPDATE)," RFC 2136, Network Working Group, ISI, April 1997.

[5] M. Ohta, "RFC 1995: Incremental Zone Transfer in DNS," internet-draft, Tokyo Institute of Technology, August 1996.

[6] J. Kangasharju and K. W. Ross, "A Replicated Architecture for the Domain Name System," in *INFOCOM (2)*, pp. 660–669, 2000.

[7] A. Tridgell and P. Mackerras, "The Rsync Algorithm," Tech. Rep. TR-CS-96-05, Australian National University, 1996.

[8] J. J. Kistler and M. Satyanarayanan, "Disconnected Operation in the Coda File System," in *Thirteenth ACM Symposium on Operating Systems Principles*,

(Asilomar Conference Center, Pacific Grove, U.S.), pp. 213–225, ACM Press, 1991.

[9] M. Satyanarayanan, "Coda: A Highly Available File System for a Distributed Workstation Environment," in *Proceedings of the Second IEEE Workshop on Workstation Operating Systems*, (Pacific Grove, CA), September 1989.

[10] 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.

[11] K. Guo, M. Hayden, R. v. Renesse, W. Vogels, and K. P. Birman, "GSGC: An Efficient Gossip-Style Garbage Collection Scheme for Scalable Reliable Multicast," tech. rep., Cornell University, December 1997.

[12] S. Agarwal, D. Starobinski, and A. Trachtenberg, "On the scalability of data synchronization protocols for pdas and mobile devices," *Special Issue of IEEE Networks on Scalability in Communication Networks*, 2002. to appear.

[13] A. Trachtenberg, D. Starobinski, and S. Agarwal, "Fast PDA Synchronization Using Characteristic Polynomial Interpolation," in *Proceedings of the IEEE Infocom*, 2002. To appear.

[14] Y. Minsky, A. Trachtenberg, and R. Zippel, "Set Reconciliation with Nearly Optimal Communication Complexity," Tech. Rep. TR2000-1813, Cornell University, 2000.

[15] A. Yaar, D. Starobinski, and A. Trachtenberg, "Engineering Scalable Data Synchronization For A Personal Digital Assistant." in preparation, 2002.

[16] A. Silberschatz, H. F. Korth, and S. Sudarshan, *Datbase System Concepts*. McGraw Hill, third ed., 1997.

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

[18] "Universal Serial Bus." http://www.usb.org.

[19] "Intellisync." http://www.pumatech.com.

[20] M. Denny and C. Wells, "EDISON: Enhanced Data Interchange Services Over Networks," May 2000. class project, UC Berkeley.

[21] D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser, "Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System," *ACM Operating System Review*, pp. 172–182, December 1995.

[22] U. Cetintemel, P. Keleher, and M. Franklin, "Support for Speculative Update Propagation and Mobility in Deno," 1999.

[23] D. Ratner, P. L. Reiher, G. J. Popek, and R. G. Guy, "Peer Replication with Selective Control," in *Proceedings of the International Conference on Mobile Data Access*, pp. 169–181, 1999.

[24] "SyncML Standard." http://www.syncml.org.

[25] M. Rabinovich, N. H. Gehani, and A. Kononov, "Scalable Update Propagation in Epidemic Replicated Databases," in *Extending Database Technology*, pp. 207–222, 1996.

[26] A. Trachtenberg, D. Starobinbki, and S. Agarwal, "Fast PDA Synchronization Using Characteristic Polynomial Interpolation," Tech. Rep. TR2001-03, Boston University, 2001.

[27] "Avantgo Mobile Internet Service." http://avantgo.com/products/ individuals/basic.html.

[28] "Infrared Data Association." http://www.irda.org.

[29] "Motorola Seminconductor Products Sector." http://e-www.motorola.com/.

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

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

[32] N. Rhodes and J. McKeehan, *Palm Programming: The Developer's Guide*. O'Reilly, first ed., 1999.

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

[34] B. S. Davie and L. L. Peterson, *Computer Networks: A Systems Approach*. Morgan Kaufman Publishers, second ed., 2000.

[35] A. S. Tanenbaum, *Computer Networks*. Prentice-Hall Inc., 3 ed., 1995.

[36] A. 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*, (Vancouver, British Columbia, Canada), pp. 1–12, ACM, August 1987.

[37] G. Apostolopoulos, R. Guerin, S. Kamat, and S. K. Tripathi, "Quality of Service Based Routing: A Performance Perspective," in *SIGCOMM*, pp. 17–28, 1998.

[38] A. Shaikh, J. Rexford, and K. G. Shin, "Evaluating the impact of stale link state on quality-of-service routing," *IEEE/ACM Transactions on Networking*, vol. 9, pp. 162–176, April 2001.

[39] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction To Algorithms*. MIT Press and McGraw-Hill Book Company, 6th ed., 1990.

[40] Y. Minsky and A. Trachtenberg, "Practical Set Reconciliation." in preparation, 2002.

Curriculum Vitae

Sachin Agarwal was born in the small south Indian town of Tumkur, on February 11, 1979, son of Rashmi Agarwal and Arun Kumar Agarwal. In 2000 he graduated with distinction from the Regional Engineering College, Warangal, India with a Bachelor of Technology degree in Electronics and Communication Engineering. At college, he was a recipient of the college merit scholarship.

He entered the graduate program in Computer Engineering offered by Boston University's College of Engineering in Fall 2000, where he also worked as a graduate research assistant in the Networking and Information Systems Laboratory. While working there, he was the co-author of the following publications

- A. Trachtenberg, D. Starobinski, and S. Agarwal, "Fast PDA Synchronization Using Characteristic Polynomial Interpolation", in the Proceedings of the IEEE INFOCOM 2002.

- S. Agarwal, D. Starobinski, and A. Trachtenberg, "On the Scalability of Data Synchronization Protocols for PDAs and Mobile Devices", special issue of the IEEE Network Magazine: Scalability in Communication Networks, July/August 2002.

He was awarded the NSF/Corporate travel award for the IEEE INFOCOM 2002 conference.

This thesis marks the end of his master's trek - uphill, but with an understanding advisor Dr. Ari Trachtenberg who offered exciting research and generous financial support, he has fond memories of all the times spent getting there. So much so that he continues to work towards his doctoral degree at Boston University, on the banks of the Charles river in Boston, Massachusetts.